Advances in Complex Systems
© World Scientific Publishing Company

ADVANCES IN DISTRIBUTED OPTIMIZATION USING PROBABILITY COLLECTIVES

DAVID H. WOLPERT
NASA Ames Research Center, Moffett Field, CA 94035
tc.arc.nasa.gov/people/dhw

CHARLIE E. M . STRAUSS
Bioscience Division, Los Alamos National Laboratory
cems@lanl.gov

DEV RAJNARAYAN
Department of Aeronautics/Astronautics
Stanford University, Stanford, CA 94305 dgorur@stanford.edu

Received (received date)
Revised (revised date)

Recent work has shown how information theory extends conventional full-rationality game theory to allow bounded rational agents. The associated mathematical framework can be used to solve distributed optimization and control problems. This is done by translating the distributed problem into an iterated game, where each agent's mixed strategy (i.e., its stochastically determined move) sets a different variable of the problem. So the expected value of the objective function of the distributed problem is determined by the joint probability distribution across the moves of the agents. The mixed strategies of the agents are updated from one game iteration to the next so as to converge on a joint distribution that optimizes that expected value of the objective function. Here a set of new techniques for this updating is presented. These and older techniques are then extended to apply to uncountable move spaces. We also present an extension of the approach to include (in)equality constraints over the underlying variables. Another contribution is that we how to extend the Monte Carlo version of the approach to cases where some agents have no Monte Carlo samples for some of their moves, and derive an "automatic annealing schedule".

Keywords: Distributed Optimization; Distributed Control; Probability Collectives.

1. Introduction

1.1. *Distributed optimization and control with Probability Collectives*

As first described in [1, 2], it turns out that one can translate many of the concepts from statistical physics, game theory, distributed optimization and distributed control into one another. This translation is based on the fact that those concepts all involve distributed systems in which the random variables are, at any single instant, statistically independent. (What is coupled is instead the distributions of those variables.) Using this translation, one can transfer theory and techniques between those fields, creating a large common mathematics that connects them. This common mathematics is known as Probability Collectives (PC). Its unifying concern is the set of probability distributions that collectively govern any particular distributed system, and how to manipulate those distributions to optimize one or more objective functions. See [3, 4] for earlier, less formal work on this topic.

In this paper we consider the use of PC to solve (potentially constrained) optimization and/or control problems. Reflecting the focus of PC on distributed systems, its use for such problems is particularly appropriate when the variables in the collective are spread across many physically separated agents with limited inter-agent communication (e.g., in a distributed design or supply chain application, or distributed control).

A crucial aspect of PC is that it concerns probabilities q over the underlying (in general multi-component) variable x, rather than that variable directly. As discussed below, this means that PC-based algorithms can be implemented for arbitrary types of (each of the components of the) underlying variable. This aspect of PC also means associated algorithms automatically provide multiple solutions to the problem.

Working with probabilities has numerous other advantages as well. Typically the q produced in a PC-based approach will be tightly peaked in certain dimensions, while being broad in other dimensions. This provides sensitivity information concerning the relative importance to solving the optimization problem of getting the values of those dimensions precisely correct. Another advantage we get by optimizing q is that we can initialize it to a set of broad peaks each centered on a solution x^i produced by some other optimization algorithm. Then as that initial q gets updated (and thereby hopefully the expected value of the objective gets improved), the set of solutions provided by those other optimization algorithms are in essence combined, to produce a solution that should be superior to any of them individually.

An advantage of PC-based approaches particularly relevant to optimization is that being inherently distributed, PC algorithms can often be implemented on a parallel computer. An advantage particularly relevant to control problems is that PC algorithms can, if desired, be used without any modelling assumptions about the (stochastic) system being controlled. Another advantage for control is that PC algorithms can work in a Monte-Carlo mode, where there is no knowledge of the functional form of the control problem. Indeed, there can be noisy components to the problem that are not governed by the PC algorithm, and the mathematics justifying the PC approach still holds.

These advantages are discussed in more detail below. First though we review PC. This review introduces a new motivation of PC, one that provides an "automatic annealing schedule". After this we introduce several new PC-based techniques, including iterative focusing. We then show explicitly how to extend PC algorithms to the case of uncountable move spaces of the players without explicitly delineating the (infinite) components of q. Some issues that arise in practice when running PC algorithms are also discussed there.

We also present an extension of PC to include (in)equality constraints over the underlying variables. Another contribution is showing how to extend the Monte Carlo version of PC to cases where some agents have no Monte Carlo samples for some of their moves.

See [5, 6, 7, 8, 9, 10, 11, 12] for other work on PC, including both software and hardware experiments. In particular, see [13, 14, 9] for work showing, respectively, how to use PC to improve Metropolis-Hastings sampling, how to relate it to the mechanism design work in [15, 4, 16, 17], and how to extend it to continuous move spaces and time-extended strategies. See [18] for a discussion of how to extend PC to higher-order graphical models than the simple ones considered here.

In [14] can be found pedagogical examples where there are closed-form solutions for the q produced by an elementary PC optimization algorithm. Also presented there is a proof that in the infinitesimal limit, many techniques for updating q become identical; these techniques all become a variant of Evolutionary Game theory's replicator dynamics in that limit. See [9] for an explicit formulation of how to apply PC to scenarios where the underlying variable is the trajectory of a multi-dimensional variable through time, i.e., to a policy-optimization scenario. Related connections between game theory, statistical physics, information theory, and PC are discussed in [2].

See [18] for discussion of the relation of PC to other probability-based approaches to optimization and control, including [19, 20, 21, 22, 23, 24, 25]. As expounded there, many of those other approaches can be seen as special cases of PC or of heuristic attempts to do what PC does formally.

1.2. *The Probability Collectives Approach*

Broadly speaking, the PC approach to solving optimization/control problems proceeds as follows. First one maps the provided problem into a multi-agent collective. In the simplest version of this process one assigns a separate agent of the collective to determine the value of each of those variables $x_i \in X_i$ in the problem that is under our control. So for example if the i'th variable can only take on a finite number of values, those $|X_i|$ possible values constitute the possible moves of the i'th agent.[a] From now on, without any loss of generality, for pedagogical simplicity we assume that all n variables are under our control. (See [8, 9].) The value of the joint set of n variables (agents) describing the system is then $x = [x_1, \cdots, x_n] \in X$ with $X \triangleq X_1 \times \cdots \times X_n$.

Unlike many optimization methods, in PC the underlying x is not manipulated directly. Rather a probability distribution over that variable is manipulated. To

[a] $|\mathcal{S}|$ denotes the number of elements in the set $\mathcal{S}$.

avoid combinatorial explosions as the number of dimensions of X grows, we must restrict attention to a low-dimensional subset of the space of all such probability distributions. We indicate this by writing the distribution over X that we manipulate as $q \in \mathcal{Q}$. The manipulation of that q proceeds through an iterative process. The ultimate goal of this process is to induce a distribution q that is highly peaked about the x optimizing the objective function of the problem. That objective function is typically written as $G(x)$, and is sometimes called the **world cost** or **world utility** function. (In this paper we only consider problems with a single overall objective function, and we arbitrarily choose lower values to be better, even when using the term "utility".) That final q is then used to determine a final answer in X, e.g., by sampling q, evaluating its mode, evaluating its mean (if that is defined), etc.

In this paper, for simplicity we consider the case where $\mathcal{Q}$ consists of all product distributions over the multi-component $x \in X$, i.e., $q(x) = \prod_{i=1}^{N} q_i(x_i)$. Most of the mathematics goes through for higher order graphical models as well, by using **semi-coordinate** transformations to express those models as product distributions in a different (potentially larger) space. In the interests of space, such transformations will not be considered here. See [9, 18] for a discussion of such transformations. In addition, none of the mathematics presented here changes if there is noise in the system. Most simply, nothing below assumes that all components of x are under our control; components of x that we do not control constitute noise. See [8].

1.3. *Advantages of Probability Collectives*

Usually during the optimization of q its support covers all of X, i.e., it lies in the interior of the unit simplices giving $\mathcal{Q}$. Conversely, for our choice of $\mathcal{Q}$, any element of X can be viewed as a delta function probability distribution, restricted to the edge (a vertex) of those simplices. So working with X is a special case of working with $\mathcal{Q}$, where one sticks to the vertices of $\mathcal{Q}$. Due to the breadth of the support of q, minimizing over it can also be viewed as a way to allow information from the value of the objective function at all $x \in X$ to be exploited simultaneously. In all this, broadening the space in which the optimization takes place to $\mathcal{Q}$ rather than X is analogous to interior point methods.

An important advantage of PC-based approaches arises from the fact that $q \in \mathcal{Q}$ is a vector with (perhaps an infinite number of) real-valued components. Due to this, finding the q optimizing the expectation value $E_q(G)$ means optimizing a real-valued function of a real-valued vector. Accordingly, using PC we can leverage the power of descent schemes for real-valued vector spaces like gradient descent or Newton's method. In particular, we can do this even if X is a categorical, finite space. So with PC, "gradient descent for categorical variables" is perfectly well-defined.

Yet another advantage, alluded to above, is that by working with distributions in $\mathcal{Q}$ rather than elements of the space X, the same general PC formalism can be used for essentially any X, be it continuous, discrete, time-extended, mixtures of these, etc. In all of these cases elements in $\mathcal{Q}$ consist of real-valued vectors. Formally, those different spaces just correspond to different probability measures, as far as PC is concerned.[b]

While prior knowledge or modelling assumptions concerning the problem can be incorporated into the PC algorithm, they are not required. Nor does the optimization require control of all components of x (i.e., some of the components of x can be noisy, can drop out of our control, etc). Furthermore, when we can sample $G(.)$ but do not know its functional form, the optimization can be done in a distributed fashion using Monte Carlo methods. All of this makes PC very broadly applicable.

The remainder of the paper is organized as follows. In section (2), we review several standard PC-based algorithms for unconstrained optimization, emphasizing their relationship with information theory and barrier function methods. We also introduce the new technique of "shrink-wrapping", which allows Monte Carlo-based techniques to be used even when some agents have unsampled moves. Next, in section (3), we introduce some variants of those standard approaches, and discuss some heuristics that have proven useful in practice. Section (4) introduces iterative focusing, a new set of PC-based techniques in which q is modified directly, rather than modified so as to optimize some functional of q. Having introduced this plethora of techniques, we present a summary of these techniques and discuss some of their relationships in section (5). Section (6) extends PC to the case where we have constraints on X. After this, we show how to extend PC to the case of uncountable X, in section (7). Finally, section (8) shows how the "trick" of subsampling allows us to modify these techniques to overome the restrictions imposed by product distributions, while still retaining the distributed nature of the algorithm.

2. Essentials of PC

For the rest of this paper we will consider PC-based algorithms concerned with finding the distribution that minimizes expected G, $\text{argmin}_{q \in Q} E_q(G)$. Other PC-based algorithms instead concern, for example, the q whose mode minimizes G, $\text{argmin}_{q \in Q} G(\max_{x \in X} q(x))$.

2.1. *The Maxent Lagrangian*

Say we are given the following problem:

(P1): Find

$$\min_{\{q_i\}} \int dx \; G(x) \prod_i q_i(x_i)$$

such that

[b]Accordingly, throughout this paper we will use the integral symbol, with the appropriate measure assumed. In particular, a point measure is implicitly assumed if the integration variable has a finite number of values, in which case an integral reduces to a sum.

$$\int dx_i \; q_i(x_i) = 1 \; \forall i$$
$$q_i(x_i) \geq 0 \; \forall i, x_i$$

where each q_i is a single-dimensional real variable.

In conventional continuous optimization one standard way to solve (P1) is by replacing the inequality constraints with **barrier functions** $\phi_i : \mathbb{R} \rightarrow \mathbb{R}$ each of which is nowhere negative and which is also infinite for negative values of its argument [26] . This replaces the inequality constrained problem (P1) with the following problem having no inequality constraints:

(P2): Find

$$\min_{\{q_i\}} [\int dx \; G(x) \prod_i q_i(x_i) + \sum_{i=1}^{N} \int dx_i \; \mu(i, x_i)\phi_i(q_i(x_i))]$$
$$\text{such that}$$
$$\int dx_i \; q_i(x_i) = 1 \; \forall i$$

where the non-negative real values $\mu(i, x_i)$ are the **barrier parameters**. Due to the nature of the barrier functions, we know that the solution to (P2) must obey the inequality constraints for any allowed values of the barrier parameters. Enforcing the equality constraints of (P2) as well guarantees that out q meets all of our constraints, i.e., is feasible.

Typically with barrier function methods one solves a sequenced of optimization problems, each of which has fixed barrier parameters. Each such problem starts with the q produced by the solution to the preceding problem. The difference between two successive problems is that the barrier parameter has been slightly reduced. As such, an annealing progresses the barrier parameters disappear and (P2) becomes (P1), i.e.,our final q is both feasible and (at least locally) solves the minimization over q of problem (P1). For convex objective functions, this process has certain guarantees of converging to the global optimum of (P1). (Note though that our objective function, $E_q(G)$, is not a convex function of the components of q; due to the fact that q is a product distribution, $E_q(G) = \int dx \prod_i q_i(x_i)G(x)$ is a multinomial function of q.)

One common choice of barrier function is $\phi(y) = y\ln(y)$ for $y > 0$, $\phi(y) = \infty$ otherwise.[c] Say we also take $\mu(i, x_i)$ to be independent of x_i, so we can write it as the vector with components $\{T_i\}$. For this case, when q is normalized (i.e., satisfies the equality constraints), the operand of the min operator becomes $E_q(G) - \sum_i T_i S(q_i)$, where $S(q)$ is the Shannon entropy of q_i. If T_i is independent of i, this difference becomes $E_q(G) - TS(q)$. This expression is called the **free energy** in statistical physics, if we identify G with the "Hamiltonian" of the system and T with its

[c]Formally, for ϕ to be a barrier function, we must add a constant large enough so that ϕ is nowhere negative, but such a constant is irrelevant for our purposes.

"temperature". See [8] for a discussion of the shape of the free energy surface for product distributions.

Let q^T be the q solving (P2) for some particular vector T. It is shown in [8] that if q consists of a single agent, i, then $E_{q^{T_i}}(G)$ is a monotonically decreasing function of T_i. In addition, typically in practice, even for multiple agents, good results arise if we anneal through a sequence of diminishing T. In light of all this, we here re-interpret (P2) as the following problem:

(P3): Find

$$\min_{\{q_i\},T}\Big[\int dx\; G(x)\prod_i q_i(x_i) + \sum_{i=1}^{N}\int dx_i\; T_i\phi(q_i(x_i))\Big]$$
$$\text{such that}$$
$$\int dx_i\; q_i(x_i) = 1\;\forall i, T \geq 0.$$

In other words, we treat T as well as q as an independent variable. Of course, we know that ultimately the solution to (P3) will have all $T_i = 0$. Our starting with a large value of the components of T simply reflects the fact that that is an easy place to start solving (P3), since it is relatively easy to solve for q^T for large T.

Continuing with our choice of entropic barrier function, we can solve (P3) via Lagrange parameters in the usual way, getting the Lagrangian

$$\mathscr{L}(q,\vec{T}) = \mathrm{E}_q(G) - \sum_i T_i S(q_i) + \sum_i \lambda_i\Big[\int dx_i\; q_i(x_i) - 1\Big] \qquad (1)$$

with the obvious extension of the definition of E and S to all of the space of real-valued possible functions over x. For simplicity from now on we will take all the T_i to be the same. In this case we refer to the expression in Eq. 1, as the **Maxent Lagrangian**. It is just the free energy plus the dot product of the Lagrange parameter vector with the equality constraint functions.

2.2. *Gradient descent of the free energy*

Say that T is fixed. In that case the entropy term in the free energy is globally convex in the remaining free variable, q. The $E_q(G)$ term in the free energy would also be convex, if we had only a single agent (in that case $E_q(G)$ would be linear in q.) Accordingly, the free energy would be convex over $\mathcal{Q}$. That is also true of the equality constraints, no matter how many agents we have. So the full Maxent Lagrangian would be convex if we had only a single agent, and the usual associated guarantees about duality gaps, saddle point solutions, etc., would apply. However for multiple agents $E_q(G)$ is not linear in q, and we do not have those convexity guarantees.

For this more general case, the simplest thing to do is an iterative descent of the free energy over $\mathcal{Q}$, where at every iteration we step in the direction within $\mathcal{Q}$

that, to first order, maximizes the drop in free energy. Since the step is restricted to lie within $\mathcal{Q}$, the equality constraints of the Maxent Lagrangian are obeyed automatically. Expanding, to first order the step that maximizes the drop in $\mathscr{L}$ is given by (-1 times)

$$\nabla_q \mathscr{L}(q) - \eta(q). \tag{2}$$

In this equation the $q_i(x_i)$ component of the gradient (one for every agent i and every possible move x_i by the agent) is

$$[\nabla_q \mathscr{L}(q)]_{q_i(x_i)} = \frac{\partial \mathscr{L}}{\partial q_i(x_i)} = E_{q_{-i}}(G \mid x_i) + T \ln[q_i(x_i)] \tag{3}$$

where

$$E_{q_{-i}}(G \mid x_i) = \int dx_{-i} q_{-i}(x_{-i}) G(x_i, x_{-i}) \tag{4}$$

with $x_{-i} \triangleq [x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n]$ and $q_{-i}(x_{-i}) \triangleq \prod_{j=1|j\neq i}^{n} q_j(x_j)$. $\eta(q)$ is the vector that needs to be added to $\nabla_q \mathscr{L}(q)$ to get it back into $\mathcal{Q}$.[d] For finite move spaces, the $q_i(x_i)$ component of $\eta(q)$ is independent of x_i, and given by

$$[\eta(q)]_i \triangleq \frac{1}{|X_i|} \int dx'_i \; [\nabla_q \mathscr{L}(q)]_{q_i(x'_i)} \tag{5}$$

where $|X_i|$ is the number of possible moves x_i. This choice ensures that $\int dx_i q_i(x_i) = 1$ after the gradient update to the values $q_i(x_i)$. The expression in Eq. 4 is the expected payoff to agent i when it plays move x_i, under the distribution q_{-i} across the moves of all other agents.

2.3. *Monte Carlo-based gradient descent and shrink-wrapping*

Eq. 2 gives the (negative of the) change that each agent should make to its distribution to have them jointly implement a step in steepest descent of the Maxent Lagrangian. These updates are completely distributed, in the sense that each agent's update at time t is independent of any other agents' update at that time. Typically at any t each agent i knows $q_i(t)$ exactly, and therefore knows $\ln[q_i(j)]$. However often it will not know G and/or the q_{-i}. In such cases it will not be able to evaluate the $E(G \mid x_i = j)$ terms in Eq. 2 in closed form.

One way to circumvent this problem is to have those expectation values be simultaneously estimated by all agents by repeated Monte Carlo sampling of q to produce a set of $(x, G(x))$ pairs. Those pairs can then be used by each agent i to estimate the values $E(G \mid x_i = j)$, and therefore how it should update its distribution. In the simplest version of this scheme such an update to q only occurs once every L time-steps. Note that only one set of Monte Carlo samples is needed for all players to determine how to update their mixed strategy, no matter how many players there are.

[d]N.b., we do *not* project onto $\mathcal{Q}$ but rather add a vector to get back to it. See [8].

In this simple Monte Carlo scheme only the samples $(x, G(x))$ formed within a block of L successive time-steps are used at the end of that block by the agents to update their distributions (according to Eq. 2). More sophisticated approaches modify the G values returned by the Monte Carlo on a player-by-player basis, etc. [1, 9].

In addition, in general it may be that some agent i has no sample for the value x_i. Some approaches to address this are presented below. These approaches are fairly elaborate, being explicitly designed to be able to work for uncountable X_i. They are based on "just in time" evaluation of how to update the probability of a move, together with interpolation to estimate quantities like $E(G \mid x_i)$.

There are a set of simpler approaches, similar to Levenberg-Marquardt descent, that are particularly appropriate for finite X_i. These approaches are known as **shrink-wrapping**. We present them here in a slightly more general context than that of minimizing the free energy.

Recall that N is the total number of agents. Expand the domain of definition of the free energy from the product of unit simplices $\mathcal{Q}$ to the Cartesian product

$$\begin{aligned}\mathcal{Q}^* &\triangleq \times_{i=1}^{N} \mathbb{R}^{|X_i|} \\ &= \mathbb{R}^{\sum_i |X_i|}.\end{aligned} \tag{6}$$

Note that $\mathcal{Q}^*$ does not contain all coupled distributions over $\mathcal{Q}$. Rather the extension is to allow the individual agents' distributions separately to be non-normalized. (However if desired the analysis presented below can be extended to allow arbitrary coupling, in a straightforward way.)

We will use the notation $u_i(x_i)$ in the obvious way, to refer to the x_i'th component of the i'th vector in the N-fold Cartesian product of vectors that we will write as the overall vector $\vec{u} \in \mathcal{Q}^*$. (An example is where the vector u is just a product distribution q, so that $u_i(x_i)$ means "$q_i(x_i)$".)

Say we are currently at a point $\vec{u} \in \mathcal{Q}^*$. Let $\vec{v}$ be an infinitesimal vector in $\mathcal{Q}^*$ proportional to the gradient of some objective function $H(\vec{u})$. As an example, we can choose to have $\vec{v}$ be an infinitesimal constant times $-E(G \mid x_i) - T\ln(q_i(x_i))$, where x_i ranges over all X_i. This is (an infinitesimal constant times the negative of) the gradient of the maxent Lagrangian over the moves of agent i, not restricted back to the unit simplex.

Write $\vec{1}$ to mean the vector in $\mathcal{Q}^*$ of all 1's, and $\vec{J}$ to mean any vector in $\mathcal{Q}^*$ all of whose components are non-negative and such that $\sum_j J_j > 0$, i.e., such that at least one component J_j is positive. Define the Hadamard product $\vec{Jv}$ by $(Jv)_j \triangleq J_j v_j \; \forall j$.

Say we take a step from the current $\vec{u}$, adding to it

$$\vec{\delta u} \triangleq \vec{Jv} - \frac{(\vec{J} \cdot \vec{v})\vec{J}}{\vec{1} \cdot \vec{J}} \tag{7}$$

Then as is trivially verified, the new $\vec{u}$ is normalized to sum to 1 if the original one is. In addition since stepping along $\vec{v}$ decreases H to first order, this alternative step does as well, decreasing it (to first order) by an amount proportional to

$$\vec{\delta u} \cdot \vec{v} = \sum_j J_j v_j^2 - \frac{(\sum_j J_j v_j)^2}{\sum_j J_j} \tag{8}$$

which is non-negative.

In particular, say $\vec{u}$ is a product distribution q, and have $\vec{v}$ be (a small constant times the negative of) the gradient of $H(\vec{u})$. Then the foregoing all holds for the choice that for all i and associated moves a_i, $J_i(a_i)$ equals 1 iff there is at least one sample of $x_i = a_i$ in our Monte Carlo sample, and 0 otherwise. With this choice we "shrink wrap" the move space of each agent i to only include those moves that occurred in the Monte Carlo block, and only perform the (normalized back to the unit simplex) gradient update on the components of q involving the moves in the shrink-wrapped space. We call this "Heaviside shrink-wrapping".

The foregoing also holds for the same $\vec{u}$ and $\vec{v}$ for many other choices of J. In particular, it holds if we take $J_i(a_i) = q_i(a_i)\ \forall i, a_i$, or the hybrid choice $J_i(a_i) = q_i(a_i)\ \forall i$ that are sampled at least once in our Monte Carlo block, 0 otherwise. The resulting rules for updating q_i are similar to the technique of "Nearest Newton", introduced blow, with the modification that we "shrink wrap" away unsampled moves.

2.4. *Automated annealing*

In the vanilla approach described above, T is fixed until q arrives at a local minimum. At that point T is reduced, via an annealing schedule, and the process repeats, with q again being optimized. [e] However in light of our formulation (P3), there is no reason not to do gradient descent in both q and T.

The partial derivative of the free energy with respect to T is just $-S(q)$. Accordingly, to first order the maximal drop in the free energy arises if the changes in T and q are related by

$$\frac{\delta q_i(x_i)}{\delta T} = \frac{\nabla_q \mathscr{L}(q) - \eta(q)}{S(q)}. \tag{9}$$

Note that by making such updates to T as we make them to q, we will typically not converge to a local equilibrium q (i.e., one for which $\nabla_q \mathscr{L} - \eta(q) = \vec{0}$) as quickly as we would if we did the descent for a fixed T. In a more conservative approach, we only update T according to Eq. 9 once we reach such an equilibrium q, and therefore typically reach such an equilibrium more quickly.

In either type of approach, the amount that T gets decreased is an increasing function of $S(q)$. Moreover, in general as the optimization progresses, q gets more peaked about the x that minimize $G(x)$, and therefore $S(q)$ shrinks. Accordingly,

[e] Indeed, simulated annealing can be viewed as a version of such descent, where rather than directly descend the free energy, one uses a Metropolis-Hastings random walk to sample the equilibrium solution where $\nabla \mathscr{L} = \vec{0}$ for the case of a single agent. See Sec. 2.5 below.

the amount that T gets decreased shrinks as the optimization progresses. This is similar to the typical (ad hoc) ways annealing schedules are set in many other optimization techniques. (Note that if each agent i had its own temperature T_i, then those T_i would get decreased by different amounts at each update; each T_i would shrink by an amount determined by the entropy of i alone.)

In practice, one might want to modify the rule Eq. 9. For example, in Monte Carlo approaches, we use an estimate of $\nabla_q \mathscr{L}$ that is necessarily somewhat noisy. (See below.) This will often mean that we anneal more slowly than might otherwise be the case. In contrast, in some scenarios we can evaluate the terms in $\nabla_q \mathscr{L}$ exactly (e.g., by having agents communicate their distributions to one another; see the KSAT experiments in [18]). In these scenarios the annealing schedule of Eq. 9 should perform better.

The Maxent Lagrangian is not unique; one can motivate other choices of what functional of q to optimize in order to minimize $E_q(G)$. Several of these other functionals of q are described below. Each of them has its own associated type of automated annealing. In addition, below we consider descent schemes based on 2nd-order expansions of the Maxent Lagrangian (i.e., based on Newton's method). Such schemes are fairly elaborate even when only descent in q is considered, with T fixed. Due to these reasons, for pedagogical simplicity here we will not consider automated annealing further; from now on we will take T to be fixed throughout epochs of varying lengths, at the end of which T is lowered according to any of the conventional semi-heuristic schemes. A full discussion of automated annealing, involving other functionals of q besides the Maxent Lagrangian, 2nd-order expansions, etc., is the subject of a paper currently in preparation.

2.5. *Brouwer updating*

It is possible to write down, explicitly, the q that forms the fixed point of gradient descent of the free energy. Setting $\tilde{\nabla}_q \mathscr{L}(q)$ to zero in Eq. 3 gives the solution

$$q_i(x_i) \propto \exp\left[-E_{q_{-i}}(G \mid x_i)/T\right] \tag{10}$$

This is a set of coupled nonlinear equations for the vectors $\{q_i\}$. Brouwer's fixed point theorem guarantees the solution of Eq. 10 exists for any G [2, 1]. Hence we call update rules based on this equation **Brouwer updating**. Note that just like gradient-descent, Brouwer updating also involves the expression $E(G \mid x_i)$. Accordingly, when that term cannot be evaluated in closed form, it can be estimated using Monte Carlo techniques, just like in gradient-descent.

In **serial** Brouwer updating, one agent at a time updates its distribution, jumping to the optimal distribution given by Eq. 10. The order in which the agents update their distributions can be pre-fixed or random. The order can also be dynamically determined in a greedy manner, by choosing which agent to update based on what associated drop in the Maxent Lagrangian would ensue [1, 14, 10]. (These various ordering schemes are similar to the those used in the majorization and block relaxation techniques in optimization statistics.) Aside from potential effects from Monte Carlo estimation error, each step of serial Brouwer updating is guaranteed not to increase the associated (Maxent) Lagrangian.

In **parallel** Brouwer updating, this updating procedure is followed simultaneously by all agents. Accordingly, when Monte Carlo sampling is used, parallel Brouwer updating can be viewed as a type of learning in games (see [21, 27, 28]). Now in general, when any q_j changes, for every $i \neq j$, what distribution q_i minimizes i's Lagrangian will change, in accord with Eq. 10. This suggests that a step of parallel Brouwer updating may "thrash", and have each agent change in a way that confounds the other agents' changes. (See [21, 28] and references therein for analysis of related issues in ficticious play.) In such a case the update may not actually decrease the associated (Maxent) Lagrangian, unlike with serial Brouwer.

There are many possible ways of addressing this problem. One is to mix parallel and serial Brouwer updating, so that only subsets of the agents perform parallel updates at any given time. These can be viewed as management hierarchies, akin to those in human organizations. Often such hierarchies can also be determined dynamically, during the updating process. Other ways to address the potential thrashing problem with parallel Brouwer have each agent i not use the current value $E_{q^t_{-i}}(G|x_i)$ alone to update $q^t_i(x_i)$, but rather use a weighted average of all values $E_{q^{t'}_{-i}}(G|x_i)$ for $t' \leq t$, with the weights shrinking the further into the past one goes. This introduces an inertia effect which helps to stabilize the updating. (Indeed, in the continuum-time limit, this weighting becomes the replicator dynamics [14].)

A similar idea is to have agent i use the current $E_{q^t_{-i}}(G|x_i)$ alone, but have it only move part of the way the parallel Brouwer update recommends. Whether one moves all the way or only part-way, what agent i is interested in is what distribution will be optimal for i *for the next distributions of the other agents*. Accordingly, it makes sense to have agent i predict, using standard time-series tools, what those future distributions will be. This amounts to predicting what the next vector of values of $E_{q^t_{-i}}(G|x_i)$ will be, based on seeing how that vector has evolved in the recent past. See [21] for related ideas.

2.6. *Kullback-Leibler based optimization*

Note that the barrier function in the Maxent Lagrangian is not continuous at the border of the feasible region. Moreover, in the language of interior point methods, it is not self-concordant. A more common choice in the literature for a barrier function is the (negative of) the logarithm, which is self-concordant. If we use that rather than the entropic barrier function we get the following Lagrangian

$$\begin{aligned} \mathscr{L}(q,\mu,\vec{T}) = \mathrm{E}_q(G) + \sum_i \lambda_i[\int dx_i \; q_i(x_i) - 1] \\ - \sum_i \int dx_i \; \mu(i,x_i)\ln(q_i(x_i))] \end{aligned} \quad (11)$$

However there are many other ways to motivate the Maxent Lagrangian [8, 9, 2] besides its derivation in terms of barrier functions. For example, information theory can be used to argue that essentially any iterative optimization algorithm can, in the absence of any prior knowledge concerning the algorithm, be modeled as descent of the Maxent Lagrangian.

Another way to motivate the Maxent Lagrangian is as the minimizer of the Kullback-Leibler distance from q to a Boltzmann distribution with Hamiltonian G and temperature T. To understand this motivation, say that we were not restricting ourselves to product distributions. So up to the additive normalizing vector the Lagrangian becomes $\mathscr{L}(p) = \beta(E_p(G) - \gamma)) - S(p)$, where p can now be any distribution over x. There is only one local minimum over p of this Lagrangian, the **canonical ensemble**:

$$p^{\beta}(x) \propto e^{-\beta G(x)}$$

In general p^{β} is not a product distribution. However we can ask what product distribution is closest to it. Now in general, the proper way to approximate a **target distribution** p with a distribution from a subset $\mathcal{C}$ of the set of all distributions is to first specify a misfit measure measuring the distance of each member of $\mathcal{C}$ to p, and then solve for the member with the smallest misfit. This is just as true when $\mathcal{C}$ is the set of all product distributions as when it is any other set.

How best to measure distances between probability distributions is a topic of ongoing controversy and research [29]. The most common way to do so is with the infinite limit log likelihood of data being generated by one distribution but misattributed to have come from the other. This is know as the **Kullback-Leibler distance** [30, 31, 32]:

$$KL(p_1 \mid\mid p_2) \triangleq S(p_1 \mid\mid p_2) - S(p_1) \tag{12}$$

where $S(p_1 \mid\mid p_2) \triangleq -\int dx\; p_1(x)\ln[\frac{p_2(x)}{\mu(x)}]$ is known as the **cross entropy** from p_1 to p_2 (and as usual we implicitly choose uniform μ). The KL distance is always non-negative, and equals zero iff its two arguments are identical.

As shorthand, define the "*pq* distance" as $KL(p \mid\mid q)$, and the "*qp* distance" as $KL(q \mid\mid p)$, where p is our target distribution and q is a product distribution. Then it is straightforward to show that the *qp* distance from q to target distribution p^{β} is just the Maxent Lagrangian $\mathscr{L}(q)$, up to irrelevant overall additive constants. In other words, the q minimizing the Maxent Lagrangian is the same as the product distribution q having minimal *qp* distance to the associated canonical ensemble.

However the *qp* distance is the (infinite limit of the negative log of) the likelihood that distribution p would attribute to data generated by distribution q. It can be argued that a better measure of how well q approximates p would be based on the likelihood that q attributes to data generated by p. This is the *pq* distance; it gives a different Lagrangian from the Maxent Lagrangian.

Evaluating, up to an overall additive constant (of the canonical distribution's entropy), the pq distance is

$$KL(p^\beta \mid\mid q) = -\sum_i \int dx\; p(x) \ln[q_i(x_i)].$$

This is rougly equivalent to a scenario where each coordinate i has its own "Lagrangian"

$$\mathscr{L}_i^*(q) \triangleq - \int dx_i\; p_i^\beta(x_i) \ln[q_i(x_i)], \tag{13}$$

where $p_i^\beta(x_i)$ is the marginal distribution $\int dx_{-i} p^\beta(x)$. Since q is a product distribution, the minimizer of the sum of the Lagrangians of each coordinate is just $q_i = p_i^\beta\ \forall i$, i.e., each q_i is set to the associated marginal distribution of p^β.

2.7. *Adaptive importance sampling*

This subsection shows how to set q to minimize pq distance. It also shows how pq distance provides a tool to perform second order descent over qp distance.

The most straightforward way to estimate the marginal distribution minimizing distance to p^β is via adaptive importance sampling, where the proposal distribution is an earlier version of q itself. In other words, first we write q in terms of an earlier estimate $\tilde{q}$ as follows:

$$\begin{aligned} q_i(x_i) &\propto \int dx'_{-i}\; \frac{e^{-\beta G(x_i, x'_{-i})}}{\tilde{q}_{-i}(x'_{-i})}\; \tilde{q}_{-i}(x'_{-i}) \\ &= \int dx'\; [\frac{e^{-\beta G(x')}}{\tilde{q}(x')}]\; [\tilde{q}(x')\delta(x_i - x'_i)] \\ &= \tilde{q}_i(x_i) \int dx' [\frac{e^{-\beta G(x')}}{\tilde{q}(x')}]\; [\frac{\tilde{q}(x)\delta(x_i - x'_i)}{\tilde{q}_i(x_i)}] \\ &= \tilde{q}_i(x_i)\; E_{\tilde{q}}(\frac{e^{-\beta G}}{\tilde{q}} \mid x_i). \end{aligned} \tag{14}$$

The exactly analogous result holds for higher-order graphical models. See [8].

To estimate the expectation value in Eq. 14 we repeatedly IID sample $\tilde{q}$, getting a set of x values. For each such x we evaluate $e^{-\beta G(x)}/\tilde{q}(x)$.[f] For each of i's possible moves, a, we then set our estimate of $q_i(x_i = a)$ to the average of the subset of those sample values of $e^{-\beta G(x)}/\tilde{q}(x)$ for which $x_i = a$. After multiplying this estimate by $\tilde{q}_i(a)$ and evaluating for all a, we renormalize those products to get a proper distribution q_i. We then set $\tilde{q}$ to that new q and iterate the process. Variants of this scheme replace the Boltzmann function $e^{-\beta G(x)}$ with a different function that is also biased toward low G values, e.g., $\Theta[K - G(x)]$. See the discussion of iterative focusing below.

[f]Note that if $\tilde{q}$ is a product distribution, this requires that with each Monte Carlo sample x each agent i broadcasts $\tilde{q}_i(x_i)$ to a central processor. That processor then forms the product of those values to evaluate $\tilde{q}$, which is broadcast back out to all the agents.

Note that the optimal solution for q_i for the pq KL Lagrangian — p_i^β — is independent of the q_{-i}. This differs from the case for the optimal solution for the qp KL Lagrangian, given in Eq. 10. However in both the adaptive importance sampling scheme for the pq KL Lagrangian outlined above, and in steepest descent for the qp KL Lagrangian, the update rule for q_i depends on the previous distributions q_{-i}. See [1].

One potential difficulty with using adaptive importance sampling this way is that it could lead to big jumps in $\mathcal{Q}$, with attendant instability arising from estimation error. An alternative approach for minimizing pq distance is to only use the adaptive importance sampling to provide the information needed to perform steepest descent on the Lagrangians $\mathscr{L}_i^*(q)$. More precisely, the steepest descent direction of that Lagrangian for agent i is given by Eq. 2, only now with the gradient term there replaced by the gradient of $\mathscr{L}_i^*(q)$, having components $-p_i^\beta(x_i)/q_i(x_i)$ (and with $\eta(q)$ redefined accordingly). The marginal $p_i^\beta(x_i)$ occuring in this gradient can be estimated via adaptive importance sampling, just as above. So with such gradient descent of $\mathscr{L}_i^*(q)$ we are still using adaptive importance sampling to estimate the marginal of p^β. However now the overall algorithm only take a small step in a direction determined by that estimate of p_i^β.

To illustrate another potential difficulty, say we have some current proposal distribution $\tilde{q}$ that is used to generate points in X. Then estimating the marginal of the Boltzmann distribution by averaging the values of $p_i^\beta/\tilde{q}$ over those points may be a very poor approach, statistically speaking. In particular, it may be that $\tilde{q}_i(x_i)p_i^\beta(x)/\tilde{q}(x) \gg 1$ for some x where $\tilde{q}_i(x_i) > 0$. In this case, with non-zero probability, until the renormalization step our estimate of $p_i^\beta(x_i)$ would be far greater than 1.

2.8. *Nearest Newton*

Care must be taken when going to second order descent methods, regardless of what Lagrangian is used. In particular, the Hessian of $\mathscr{L}_i^*(q)$ is diagonal and therefore trivial to invert. It is also positive-definite. However applying a Newton-Raphson step with that Hessian, for example, would result in a new q that doesn't lie on $\mathcal{Q}$. Moreover, such an update step is independent of p^β. Such problems are even more acute when trying to descend the Maxent Lagrangian, since coupling between the agents in the multilinear term $E(G)$ makes the associated Hessian non-diagonal.

An alternative approach for second order descent of the Maxent Lagrangian starts by making a quadratic approximation (over the space of all distributions p, not just all product distributions q) to the Lagrangian, $\mathscr{L}(p)$ based on the current point p^t. Newton's method then specifies a p^{t+1} that minimizes that quadratic approximation. We can then find the product distribution that is nearest (in pq KL distance) to p^{t+1} and move to that product distribution. The resultant update rule for the Maxent Lagrangian is called **Nearest Newton** descent [8]:

$$\frac{q_i^{t+1}(j)}{q_i^t(j)} = 1 - S(q_i^t) - \ln(q_i^t(j)) \\ - \beta[E_{q^t}(G \mid x_i = j) - E_{q^t}(G)] \tag{15}$$

where q^t is the current (assumed to be product) distribution. The conditional expectations in Nearest Newton are the same as those in gradient descent. Accordingly, they too can be estimated via Monte Carlo sampling, if need be.

In [14] it is shown that in the continuum time limit, Nearest Newton updating for modifying the probability distribution of any particular agent becomes a variant of replicator dynamics (with the different strategies of replicator dynamics identified with the different possible moves of the agent performing the Nearest Newton update). That paper also shows that that when the terms in that continuum limit version of Nearest Newton are Monte-Carlo estimated, Nearest Newton becomes a version of ficticious play. More precisely, it becomes identical to a "data-aged" continuum time limit of parallel Brouwer updating. The stepsize of the Nearest Newton procedure is identical to the constant in the exponent of the data-aging.

3. Variants of Standard PC Approaches and Heuristics

Aside from shrink-wrapping, all the schemes considered above have been investigated in experiments. This section presents variants of those schemes that are yet to be investigated in experiments, as well as some heuristics that have proven useful in practice.

3.1. *Modifications to the Monte Carlo Process of Parallel Brouwer*

As described above and in [9], parallel Brouwer updating can be subject to "thrashing", in which each player's update confounds the updates of the other players. The simplest way to mitigate this is by not having each player i jump all the way from its current distribution q_i to the new one recommended by parallel Brouwer updating, q_i^*. Instead one can have each i only jump part way in the direction from q_i to q_i^*. (This in fact is what is done in practice.) Another common way to mitigate thrashing is to use data-aging. In this approach each conditional expectation value in an update rule is replaced by a decaying average of its previous values. This subsection presents a third approach.

To begin, note that we would not get any thrashing in parallel Brouwer if rather than the function $E_q(G \mid x_i)$, each agent i performed its update using $E_\pi(G \mid x_i)$ for some fixed distribution π that is independent of both i and q. The natural choice of π is exactly the distribution that q is designed to approximate well, namely the Boltzmann distribution.[g]

[g]Note that in doing this, we change the equilibrium distribution from that of Eq. 10. Now it is given by $q_i(x_i) \propto e^{-\beta E_\pi(G|x_i)}$

To implement this modification to parallel Brouwer, we need to have all agents i simultaneously estimate their associated functions $E_\pi(G \mid x_i)$ rather than $E_q(G \mid x_i)$. Precisely because q should approximate π well, we can do this using our Monte Carlo samples of q, simply by modifying how each agent uses those samples. The general idea is to use those samples of q as a proposal distribution for generating samples from π.

As an example, we can use the samples of q to estimate the integral $E_\pi(G \mid x_i)$ via importance sampling. To do this we write

$$E_\pi(G \mid x_i) = \frac{\int dx'_{-i}\, [\frac{\pi(x_i, x'_{-i})}{q(x_i, x'_{-i})} G(x_i, x'_{-i})] q(x_i, x'_{-i})}{\int dx'_{-i}\, [\frac{\pi(x_i, x'_{-i})}{q(x_i, x'_{-i})}] q(x_i, x'_{-i})},$$

and then sample q, using empirical averages across those samples to estimate both the quantity in the square brackets in the numerator of our integral and the quantity in square brackets in the denominator. (Note that we only need to know π up to an overall normalization constant to do this.) Under the original sampling scheme, for each of its possible moves x_i, agent i forms the uniform average of the G values that arose when it made that move, and takes that average as its estimate of $E_q(G \mid x_i)$. Under the modified scheme, it would instead estimate the function $E_\pi(G \mid x_i)$ with a weighted average of those G values. The weights would be the associated values $\pi(x)/q(x)$.[h]

Another way to estimate $E_\pi(G \mid x_i)$ using samples generated from q would be via a Metropolis random walk. Under this scheme q would be a proposal distribution, and the points it generates would be kept either if they raised $\pi(x)$, or, if not, if the flip of an appropriately weighted coin comes up heads. At the end of the Monte Carlo block, each agent i would form the uniform averages over the kept points, thereby forming an estimate of its function $E_\pi(G \mid x_i)$ [i].

This particular integration of parallel Brouwer and the Metropolis-Hastings algorithm can be motivated other ways than as a modification to parallel Brouwer updating. In particular, it can be motivated as a modification to the standard optimization algorithm of simulated annealing.

To see this, recall that in standard simulated annealing with a product proposal distribution the sample values of each coordinate are chosen in a non-adaptive, fixed manner, without regard to the results of previous sampling. An alternative would be to have a Reinforcement Learning (RL) [33] agent associated with each coordinate, and have each such agent choose the sample values of the coordinate it controls. By giving the agents rewards based on values of $G(x)$, this should result

[h]Note that these weights can be communicated to all the agents by the same system that broadcasts G values to all the agents, if first all agents communicate q_i values to that system.

[i]Ref. [13] presents a detailed analysis of the use of samples of a product distribution to do Metropolis-Hastings sampling. That work does not directly concern the issue of optimization. Rather it concentrates on using Probability Collectives to improve the usual goal of the Metropolis-Hastings algorithm, namely sampling a provided probability distribution.

in "intelligently" chosen sample points. This is in contrast to the situation (as in the conventional simulated annealing algorithm) where the sampling distribution is pre-fixed and "dumb".

Now one common RL algorithm has its agent sample from its possible moves according to a Boltzmann distribution across its expected rewards for those moves. Let those rewards be the values of $G(x)$. Then this common RL algorithm for an "intelligent" agent has that agent use the Brouwer updating algorithm to update its distribution, and then samples from that distribution. This is algorithmically identical to the scheme discussed above for "integrating parallel Brouwer and the Metropolis-Hastings algorithm". It is the basis of the Intelligent Coordinates algorithm which experimentally appears to far outperform simulated annealing [3].

3.2. *Variants of Maxent Lagrangians*

Consider the use of iterative update rules for the q_i in concert with Monte Carlo sampling of q. In such scenarios, at each stage of the iterative updating, for each of her moves x_i, each player i has an empirical estimate of the distribution $P(G \mid x_i)$ (and therefore of any distribution $P(f(G) \mid x_i)$ for invertible $f : \mathbb{R} \rightarrow \mathbb{R}$). Every player i uses her empirical estimate according to a pre-set algorithm — potentially varying from one player to the next — to determine how to update her distribution q_i. Our task as system designers is to choose those pre-set algorithms in such a way that the ultimate goal of the updating is achieved as quickly as possible.

In the update rules discussed above each empirical distribution is reduced to an expectation value which is then used to perform the update. While this need not be the case in general, update rules based on expectation values form a very rich set, including many rules not investigated previously. This subsection introduces some such novel update rules that are based on expectation values.

Both the qp-KL Lagrangian and pq-KL Lagrangians discussed above had the target distribution be a Boltzmann distribution over G. For high enough β, such a distribution is peaked near $\text{argmin}_x G(x)$. So sampling an accurate approximation to it should give an x with low G, if β is large enough. This is why one way to minimize G is to iteratively find a q that approximates the Boltzmann distribution, for higher and higher β.

However there are other target distributions that grow larger as G grows smaller e.g., logistic functions of G, step functions (i.e., Heaviside functions) of G, etc. So one set of alternatives to the Lagrangians discussed above is to choose some alternative target distribution(s), and for each one find the q minimizing pq or qp KL distance to it.

Return now to the Maxent Lagrangian. Say that after finding the q that minimizes the Lagrangian, we IID sample that q, K times. We then take the sample that has the smallest G value as our guess for the x that minimizes $G(x)$. For this to give a low x we don't need the mean of the distribution $q(G)$ to be low — what we need is for the bottom tail of that distribution to be low. This suggests that in the $E(G)$ term of the Maxent Lagrangian we replace

$$q(x) \leftarrow q(x)\frac{\Theta[\kappa - \int dx'\ q(x')\Theta[G(x) - G(x')]]}{\kappa}. \tag{16}$$

The new q for $E_q(G)$ given by Eq. 16 is still a probability distribution over x. It equals 0 if $G(x)$ is in the worst $1 - \kappa$ percentile (according to distribution q) of G values, and κ^{-1} otherwise. So under this replacement the $E(G)$ term in the Lagrangian equals the average of G restricted to that lower κ'th percentile. For $\kappa = K^{-1}$, our new Lagrangian forces attention in setting q on that outlier likely to come out of the K-fold sampling of $q(G)$.[j]

As usual, one can use gradient descent and Monte Carlo sampling to minimize this Lagrangian, taking care to account for q's now appearing twice in the integrand of the $E(G)$ term. Note that the Monte Carlo process includes sampling the probability distribution $\frac{\Theta[\kappa - \int dx'\ q(x')\Theta[G(x)-G(x')]]}{\kappa}$ as well as the q_i. This means that only those points in the best κ'th percentile are kept, and used for all Monte Carlo estimates. This may cause greater noise in the Monte Carlo sampling than would be the case for $\kappa = 1$.

As an example, say that for agent i, all of its moves have the same value of $E(G \mid x_i)$, and similarly for agent j, and say that G is optimal if agents i and j both make move 0. Then if we modify the updating so that agent i only considers the best values that arose when it made move 0, and similarly for agent j, then both will be steered to prefer to make move 0 to their alternatives. This will cause them to coordinate their moves in a way that improves the Lagrangian.

A similar modification is to replace G with $f(G)$ in the Maxent Lagrangian, for some monotonically increasing function $f(.)$. This would distort G to accentuate those x's with good values. Intuitively, this will have the effect of coordinating the updates of the separate q_i at the end of the block, in a way to help lower G. The price paid for this is that there may be more variance in the values of $f(G)$ returned by the Monte Carlo sampling than those of G, in general.[k]

Note that if q is a local minimum of the Lagrangian for G, in general it will not be a local minimum for the Lagrangian of $f(G)$ (the gradient will no longer be zero under that replacement, in general). So we can replace G with $f(G)$ when we get stuck in a local minimum, and then return to G once q gets away from that local minimum. In this way we can break out of local minima, without facing the penalty of extra variance. Of course, none of these advantages in replacing G with

[j]This algorithm should be contrasted to iterative focusing, where (in one version) we solve for the new distribution closest to the q given by Eq. 16, whereas here we directly insert that new q into the $E_q(G)$ component of the Maxent Lagrangian.

[k]Write γ for the value of $E(G)$ at the moment we replace $G \rightarrow f(G)$. Then it may make sense to require that $E(G) \leq \gamma$ even after the replacement. This could be done in the usual way by adding a term $\alpha[E(G) - \gamma]$ to the Lagrangian. The Lagrange parameter α would initially equal 0, and then get updated by gradient ascent on the Lagrangian periodically. So it would periodically get increased by an amount proportional to the violation factor $E(G) - \gamma$, thereby "annealing in" our constraint.

$f(G)$ hold for algorithms that directly search for an x giving a good $G(x)$ value; x is a local minimum of $G(x) \Leftrightarrow x$ is a local minimum of $f(G(x))$.

An even simpler modification to the $E(G)$ term than those considered above is to replace $G(x)$ with $\Theta[G(x) - K]$. Under this replacement the $E(G)$ term becomes the probability that $G(x) > K$. So minimizing it will push q to x with lower G values. For this modified Lagrangian, the gradient descent update step adds the following to each $q_i(x_i)$:

$$\alpha\big[\beta q(G < K \mid x_i) + \ln(q_i(x_i)) \;-\; \frac{\sum_{x'_i} \beta q(G < K \mid x'_i) + \ln(q_i(x'_i))}{\sum_{x'_i} 1}\big]. \tag{17}$$

In gradient descent of the Maxent Lagrangian we must Monte Carlo estimate the expected value of a real number (G). In contrast, in gradient descent of this modified Lagrangian we Monte Carlo estimate the expected value of a single bit: whether G exceeds K. Accordingly, the noise in the Monte Carlo estimation for this modified Lagrangian is usually far smaller. In addition, just like in descent of the Maxent Lagrangian, the Monte Carlo estimation for Eq. 17 is well-suited to a distributed implementation.

In all these variants it may make sense to replace the Heaviside function with a logistic function or an exponential. In addition, in all of them the annealing schedule for K can be set by periodically searching for the K that is (estimated to be) optimal, just as one searches for optimal coordinate systems [2, 1]. Alternatively, a simple heuristic is to have K at the end of each block be set so that some pre-fixed percentage of the sampled points in the block go into our calculation of how to update q.

Yet another possibility is to replace $E(G)$ with the κ'th percentile G value, i.e., with the K such that $\int dx'\, q(x')\Theta(G(x') - K) = \kappa$. (To evaluate the partial derivative of that K with respect a particular $q_i(x_i)$ one must use implicit differentiation.)

3.3. *Heuristics for improving the update rules*

There are a number of practical issues arising in many of the schemes elaborated above. The update rules given above are all completely distributed, in the sense that each agent's update at time t is independent of any other agents' update at that time. Typically at any t each agent i knows $q_i(t)$ exactly, and therefore knows $\ln[q_i(j)]$. However many of those update rules for each agent i involve conditional expectation values dependent on q_{-i}. Moreover in practice often those expectation values cannot be evaluated in closed form. As described above though, one can circumvent this problem by having the expectation values be simultaneously estimated by all agents via repeated Monte Carlo sampling of q to produce a set of $(x, G(x))$ pairs. Those pairs are used by each agent i to estimate the expectation values it needs (e.g., $E(G \mid x_i = j)$), and therefore how to update its distribution.

Consider the case where we do need to use Monte Carlo to estimate conditional expected values of some $f(x)$, and x is high-dimensional. In this scenario block-wise Monte Carlo sampling to estimate conditional expectation values can be slow. The estimates typically have high variance, and therefore require large block size L to get an accurate estimate.

One set of ways to address this is to replace the team game with a non-team game, i.e., for each agent i have it estimate quantities based on a **private utility** g_i rather than G (e.g., based on $E(g_i \mid x_i = j)$ rather than $E(G \mid x_i = j)$ [l]. Each such private utility is chosen so that the Monte Carlo estimates have much lower variance than those based on G, without having any bias [1, 14].

As an example, say we are doing gradient descent of the Maxent Lagrangian. Replace the values of $G(x)$ recorded by agent i in the Monte Carlo process with the values of $g_i(x) = G(x) - D(x_{-i})$, where $D(x_{-i}) \propto \int dx'_i \; w(x'_i)G(x'_i, x_{-i})$ for weighting factors w_i determined by how frequently x'_i arose in the Monte Carlo process. This replacement speeds the convergence of the Monte Carlo process to accurate estimates of the true expectation values $E(G \mid x_i)$ [1]. Furthermore it can often be done with minimal communication overhead between the agents. Indeed, often it is easier to evaluate such a $g_i(x)$ than $G(x)$. The worst case is where $G(x'_i, x_{-i})$ must be explicitly re-evaluated for each of the possible x'_i. Even there though, those extra re-evaluations are often not a large extra expense. This is because they can be used to augment the Monte Carlo samples of values of $g_i(x^*_i)$ for $x^*_i \neq x_i$ as well as those for $x^*_i = x_i$.

Another useful technique is to allow samples from preceding blocks to be re-used. One does this by first "aging" that data to reflect the fact that it was formed under a different q_{-i} . For example, one can replace the empirical average for the most recent block k,

$$\hat{G}_{i,j}(k) \equiv \frac{\sum_{t=kL}^{kL+L} G(x^t)\delta_{x_i^t,j}}{\sum_{t=kL}^{kL+L} \delta_{x_i^t,j}},$$

with a weighted average of previous expected G's,

$$\frac{\sum_m \hat{G}_{i,j}(m)e^{-\kappa(k-m)}}{\sum_m e^{-\kappa(k-m)}}$$

for some appropriate aging constant κ.[m]

[l]Formally, this means that each agent i has a separate Lagrangian, for example formed from the Maxent Lagrangian by substituting g_i for G. See [2] for the relation of this to bounded rational game theory.

[m]Not all preceding $\hat{G}_{i,j}(m)$ need to be stored to implement this; exponential ageing can be done online using 3 variables per (i, j) pair. Say agent i has just made a particular move, getting cost r, and that the most recent previous time it made that time was T iterations ago. Then the new estimated cost for that move, E', is related to the previous one, E, by $E' = \frac{r+k^T Ea}{1+k^T a}$, where k is a constant less than 1, and a is initially set to 1, while itself also being updated according to a += k^T. So agent i only needs to keep a running tally of E, a, and T for each of its possible moves to use data-aging, rather than a tally of all historical time-cost pairs.

Typically such ageing allows L to be vastly reduced, and therefore the overall optimization of $\mathscr{L}$ to be greatly sped up. For such small L though, it may be that the most recent block has *no* samples of some move $x_i = j$. This would mean that $\hat{G}_{i,j}(k)$ is undefined. One crude way to avoid such problems is by simply forcing a set of samples of each such move if they don't occur of their own accord, being careful to have the x_{-i} formed by sampling q_{-i} when forming those forced samples. A more sophisticated approach is to use shrink-wrapping, introduced above. Another approach involves supervised learning, as described in the section on "Empty bins" below.

There are numerous other techniques that are useful in practice. For example, typically one must use such techniques to decrease the step size in the descent rules (i.e., gradient descent and Nearest Newton) as one nears the border of $\mathcal{Q}$. Similarly, often the non-descent update rules (e.g., Brouwer) can be improved by making only a partial "step" at each iteration, i.e., by averaging the current q with the q given by the update rule as listed above, rather than by replacing it with that q.

Note that these practical difficulties do not apply to all updating schemes. In particular, in adaptive importance sampling, each agent i's update rule has no direct dependence on q_{-i}. (The only dependence is indirect, via the fact that the generation of the Monte Carlo samples uses $\tilde{q}_i$; there is no expression like $E_{q_{-i}}(G \mid x_i)$ in the update rule.) Accordingly, *all* previous Monte Carlo data can be used, without any data-aging or the like. All that is required is that the values $G(x)$ and $\tilde{q}_{-i}(x)$ — which each agent uses to perform its update at the time the Monte Carlo sample x is generated — are recorded for later use. Given those recorded values, at any update, the agent simply averages the ratios $\exp(-\beta G(x))/\tilde{q}_{-i}(x)$ for all preceding data points, regardless of whether some of them were made with different distributions $\tilde{q}_{-i}(x)$ from other ones. The normalized version of those averages gives its updated estimate of the marginal of the Boltzmann distribution p^β, just like normal.

In fact, given the recorded values $G(x)$ and $\tilde{q}_{-i}(x)$, it is even possible to re-use old data after β has changed, if one has evaluated and recorded the normalization constant of p^β for those old β values. Furthermore, since there is no need for an agent i to estimate a functional of q_{-i}, there is not even a need for Monte Carlo "blocks" of length L, per se. q can be updated continually after each new Monte Carlo sample, rather than only at the end of such a block of L samples.

4. Iterative Focusing

This section introduces a class of schemes for minimizing $E_q(G)$, without direct concern for the Maxent Lagrangian.

4.1. *Iterative focusing for pq KL distance*

Given some current distribution, often one can generate distributions that will be more peaked about low $G(x)$ than that current distribution by appropriately transforming that current distribution. This suggests a ratchet-like process that iterates a two-step procedure: First one applies such a transformation to the current product distribution, generating a new distributions that in general need not be a product. Then one finds the q that best approximates that transformed version, to get the product distribution for the next iteration. More concretely, the process of **iterative focusing** proceeds as follows:

1) Based on G, choose an associated **focusing operator** $\mathcal{M} : \mathcal{P} \rightarrow \mathcal{P}$ such that $E_{\mathcal{M}(p)}(G) < E_p(G)$ for any distribution p. In general $\mathcal{M}(p) \notin \mathcal{Q}$. So we also choose a distance measure $\mathcal{D}(.,.)$ across $\mathcal{P}$, which we will use to get back to $\mathcal{Q}$.

2) Given a product distribution $\tilde{q}$, solve for $\text{argmin}_{q \in \mathcal{Q}} \mathcal{D}(q, \mathcal{M}(\tilde{q}))$. (In general, the finding of that minimizing q may be an extensive multi-step process in its own right.)

3) Set the new $\tilde{q}$ to that solution from (2). Potentially "anneal" $\mathcal{M}$ as well, to tighten the focusing around x with lower values of $G(x)$. Return to (2).

We would like to be able to guarantee that in each pass iterative focusing produces a distribution that is closer to a delta function about $\text{argmin}_x G(x)$. If $\mathcal{M}(\tilde{q})$ were $\in \mathcal{Q}$, so that $\mathcal{D}$ were unnecessary, we would have such a guarantee. However when this is not the case, the distribution output in step (3) $\neq \mathcal{M}(\tilde{q})$, and it may not be superior to the distribution input to step (2). Ultimately whether the distribution produced by step (3) is superior to the one input to step (2) depends on the relation of $\mathcal{D}$, $\mathcal{Q}$, the $\tilde{q}$ input to step (2), and $\mathcal{M}(.)$.

The simplest choice of focusing operator is multiplication by an a nowhere-negative focusing function $F_G(x)$ that increases as $G(x)$ decreases, followed by renormalization:

Proposition 1: Say F_G is integrable and nowhere-negative and $G(x') \leq G(x) \Rightarrow F_G(x') \geq F_G(x)\ \forall x, x'$. Then F_G is a single-valued function of $G(x)$, and in addition the focusing property of step (1) is guaranteed for $\mathcal{M}(p)(x) = \frac{F_G(x)p(x)}{\int dx\ F_G(x)p(x)}$.

Proof: Write $F_G(x) = f(G(x))$ and let y indicate a generic value of $G(x)$. Then the post-focusing distribution over y is

$$P(y) = \int dx \delta(G(x) - y) \frac{f(G(x))p(x)}{\int dx\ f(G(x))p(x)} \ = \ \frac{f(y)p(y)}{\int dy\ f(y)p(y)}$$

where $p(y) = \int dx\ p(x)\delta(y - G(x))$. Define the distribution $r(y) \triangleq \frac{f(y)p(y)}{\int dy f(y)p(y)}$. Then

$$\frac{r(y)}{p(y)} \geq 1 \Rightarrow \forall y' \leq y, \frac{r(y')}{p(y')} \geq 1,$$
$$\frac{r(y)}{p(y)} \leq 1 \Rightarrow \forall y' \geq y, \frac{r(y')}{p(y')} \leq 1.$$

Therefore there must be a greatest value a such that $\forall y \leq a$, $\frac{r(y)}{p(y)} \geq 1$, and a smallest value b such that $\forall y \geq b$, $\frac{r(y)}{p(y)} \leq 1$. Now by normalization,

$$\int_{-\infty}^{a} dy[r(y) - p(y)] = -\int_{b}^{\infty} dy[r(y) - p(y)],$$

and $r(y) = p(y)$ throughout the range (a, b). Write the change in expected G when multiplying $p(x)$ by $F_G(x)$ and renormalizing as

$$\begin{aligned}\Delta(G) &= \int_{-\infty}^{\infty} dy\; y[r(y) - p(y)] \\ &= \int_{-\infty}^{a} dy\; y[r(y) - p(y)] + \int_{b}^{\infty} dy\; y[r(y) - p(y)].\end{aligned}$$

The first integral is bounded above by $a \int_{-\infty}^{a} dy[r(y) - p(y)]$, and the second is bounded above by $b \int_{b}^{\infty} dy[r(y) - p(y)]$. Combining gives $\Delta(G) \leq 0$. **QED.**

Without loss of generality, we can take $F_G(x)$ to be a probability distribution.

As any example, given any distribution p, the focused distribution

$$\mathcal{M}_{\Theta(K-G(.))}(p)(x) \triangleq \frac{p(x)\Theta[K - G(x)]}{\int dx'\; p(x')\Theta[K - G(x')]}$$

is guaranteed to be more peaked about x with small $G(x)$ than is p. So we can minimize $G(x)$ by iterating the process of finding the product distribution q that best approximates $\mathcal{M}_{\Theta(K-G(.))}(p)$ and then setting $p = q$. In doing this we can also gradually decrease K, to get distributions that are more and more restricted to $\text{argmin}_x G(x)$. This "annealing" is analogous to an iteration of the process of finding the q that best approximates the Boltzmann distribution for a particular β and then increasing β.

It would be nice to choose a distance measure $\mathcal{D}$ to minimize the chances that the projection back into $\mathcal{Q}$ needed in iterative focusing thwarts the improvement given by applying the focusing operator $\mathcal{M}$. However it's not clear how best to do that. So as an alternative, here we focus on the simplest choices of distance measure, the *pq* and *qp* KL distances.

Continuing with our example, for the choice of *pq*-KL distance as the distance measure $\mathcal{D}$, the q that best approximates $\mathcal{M}_{\Theta(K-G(.))}(p)$ is just the product of the marginal distributions of $\mathcal{M}_{\Theta(K-G(.))}(p)$. So in each pass of the associated iterative focusing algorithm,

$$\begin{aligned}q_i(x_i) \leftarrow &\frac{\int dx'_{-i} q(x'_{-i}, x_i)\Theta[K - G(x_i, x'_{-i})]}{\int dx'\; q(x')\Theta[K - G(x')]} \\ &= \frac{q(G < K, x_i)}{q(G < K)} \\ &= q(x_i \mid G < K) \qquad (18) \\ &\propto q(G < K \mid x_i) q_i(x_i) \qquad (19)\end{aligned}$$

This new q_i can be Monte-Carlo estimated by agent i using only observed G values, in the usual way. So like gradient descent on the Maxent Lagrangian, this update rule is well-suited to a distributed implementation. Indeed, the only term that needs Monte Carlo estimating is $q(G < K \mid x_i)$.

4.2. *Iterative focusing and Brouwer updating*

One obvious potential problem with this updating rule is that the randomness of the Monte Carlo process might erroneously lead one to set $q_i(x_i = a)$ to 0 even though the x that minimizes G has its i'th component equal to a. Once this happens recovery is impossible; there is no way for $q_i(x_i = a)$ to increase from 0 subsequently.

We can avoid this problem if we replace the Heaviside focusing function with a "softened version" like a logistic function with exponent β about K, $\tilde{\Theta}_{\beta,K}(x) \triangleq [1 + e^{\beta(G(x)-K)}]^{-1}$. With this change the update rule becomes

$$q_i(x_i) \leftarrow \frac{E(\tilde{\Theta}_{\beta,K} \mid x_i)q_i(x_i)}{E(\tilde{\Theta}_{\beta,K})}. \tag{20}$$

As another alternative, we can replace the Heaviside function with a Boltzmann distribution with exponent β, getting the update rule

$$q_i(x_i) \leftarrow \frac{E(e^{-\beta G} \mid x_i)q_i(x_i)}{E(e^{-\beta G})}. \tag{21}$$

where all terms on the righthand side are evaluated under the distribution that generated the Monte Carlo samples.

Unlike the update rule of Eq. 19 or Eq. 20, with Eq. 21 we don't need to specify an annealing schedule under which the focusing function changes as the algorithm progresses. The annealing is automatic, due to the fact that multiplying by $e^{-\beta G(x)}$ at each iteration is the same as increasing β by the same constant at each iteration, i.e., due to the fact that the Boltzmann distribution is a Lie group with parameter β.

The update rule of Eq. 21 is similar to that of Eq. 14, only here to form the quantity being averaged one does not divide by $q(x)$. It's also very similar to the **Brouwer update rule** applied to the team game [10, 1, 34]. However in contrast to the potential "thrashing" of parallel Brouwer updating, the update in Eq. 21 is guaranteed to minimize its associated Lagrangian of pq distance to $\mathcal{M}_{\Theta(K-G(.))}(q)$ (assuming no error in estimating the expectation values). On the other hand, in Eq. 21 the Lagrangian is not distance to a fixed distribution, as it is with the Maxent Lagrangian of parallel Brouwer updating and with the update of Eq. 14. Rather the "target distribution" itself changes from one iteration to the next. This is actually the case for all iterative focusing, and is the analogue of the thrashing of parallel Brouwer updating.

4.3. *Iterative focusing for qp distance*

Rather than *pq* KL distance, consider using *qp* KL distance as the error measure for approximating the Boltzmann-focused distribution, $\tilde{q}(x)\frac{e^{-\beta G(x)}}{\int dx' \; \tilde{q}(x')e^{-\beta G(x')}}$. Up to an overall additive constant, that distance is just the Maxent Lagrangian, with one difference: we replace the $S(q)$ term with $-KL(q \;\; || \; \tilde{q})$. This suggests an iterative algorithm which starts with a distribution $\tilde{q}$, and then searches for the q minimizing a modified version of the Maxent Lagrangian $\mathscr{L}(q)$ in which the $S(q)$ term is replaced by $-KL(q \;\; || \; \tilde{q})$. When that search terminates one resets $\tilde{q}$ to that minimizing q and starts all over again.

The natural choice for the initial $\tilde{q}$ is the uniform distribution. In this case $-KL(q \;\; || \; \tilde{q}) = S(q)$, so our KL distance for the first pass of the algorithm reduces to the usual form of the Maxent Lagrangian. Therefore the first stage of the algorithm proceeds to a local minimum q^1 of the Maxent Lagrangian (via Nearest Newton, Brouwer updating, or some such). Once that local minimum is found, there are many schemes to break out of it. One of the most obvious is to simply raise (!) β and restart the descent. Such a change to β is equivalent to modifying our current target distribution, $p \triangleq \tilde{q}\frac{e^{-\beta G(x)}}{\int dx' \; \tilde{q}(x')e^{-\beta G(x')}}$, by multiplying it by another Boltzmann distribution (and then renormalizing).

The most natural choice for when to change $\tilde{q}$ to q^1 is after one has gone past all such local minima to a global minimum (or more generally, to what one hopes is a global minimum). However there are many alternatives. For example, one could do the replacement at the first local minimum. As opposed to multiplying the current target p by a Boltzmann distribution, such a replacement of $\tilde{q}$ would make a new target distribution by multiplying q^1 by a Boltzmann distribution. Obvious variants of this scheme weave the resetting of the target p more frequently into the overall process, so that it is updated before a local minimum is found. Indeed, one can even have the target reset after every modification of q, to be the preceding q.

As an example of the foregoing, given the current product distribution $\tilde{q}$, the optimal solution Eq. 10 changes to

$$q_i^\beta(x_i) \propto \tilde{q}_i(x_i)e^{-\beta E_{q^\beta_{-i}}(G|x_i)}$$

So Brouwer updating is now different from what it is in the conventional Maxent Lagrangian case, with the distribution $\tilde{q}$ serving as a prior probability μ (recall the Maxent Lagrangian). If at each $t+1$ we update $\tilde{q}^{t+1}$ to q^t, and (as in conventional Brouwer) use q^t to estimate the expectation value, the update rule is

$$q^{t+1}(x_i) \propto q_i^t(x_i)e^{-\beta E_{q^t}(G|x_i)}. \tag{22}$$

Just as the parallel, serial, etc., variants of Brouwer updating all have their strengths, so there are reasonable schemes for how to set the updating of the distributions on the righthand side of Eq. 22. For example, one might replace the $q_i^t(x_i)$ term on the righthand side of Eq. 22 with $q_i^{t'}(x_i)$ for some earlier t'. Another possibility is to replace the q^t in the exponential on the righthand side with $q^{t''}$ for some

$t'' < t$, and more generally one can vary which of these replacements gets made when.

It is interesting to compare the variant of parallel Brouwer given by Eq. 22 with Nearest Newton applied to the Maxent Lagrangian. If we expand the exponential in Eq. 22 to first order we get

$$q_i^{t+1}(x_i) \propto q_i^t(x_i)[1 - \beta E_{q^t}(G \mid x_i)]. \tag{23}$$

If we now approximate the update of Nearest Newton by removing the ln term (i.e., by taking temperature to 0, with stepsize changed accordingly), we get an update rule almost identical to that of Eq. 23. (The remaining difference is that Nearest Newton normalizes the update to stay in $\mathcal{P}$ by adding a normalizing vector rather than by dividing by a normalizing scalar.) This connection is not too surprising, in light of the fact that in the continuum time limit with data-aging, Nearest Newton and parallel Brouwer updating become identical, with the stepsize of the Nearest Newton identically equal to the data aging-constant in the parallel Brouwer [14].

In addition to Brouwer updating, gradient descent also changes when we use it for iterative focusing of $\tilde{q}$ using qp distance rather than for minimization of the Maxent Lagrangian. The term $u_i(j)$ of Eq. 2 that sets the descent direction becomes

$$\beta E(G \mid x_i = j) + \ln[\frac{q_i(x_i)}{\tilde{q}_i(x_i)}]. \tag{24}$$

So by iteratively focusing $\tilde{q}$ rather than descending the Maxent Lagrangian we penalize q's that differ from $\tilde{q}$. If $\tilde{q}$ is updated frequently, this provides an inertia effect in the dynamics of q, impeding it from changing too fast.[n]

Nearest Newton also changes when used for iterative focusing rather than descending the Maxent Lagrangian. It becomes

$$\begin{aligned}\frac{q_i^{t+1}(j)}{q_i^t(j)} &= 1 + KL(q_i^t||\tilde{q}_i) - q_i^t(j)\ln[\frac{q_i^t(j)}{\tilde{q}_i(j)}] \\ &\quad - \beta[E_{q^t}(G \mid x_i = j) - E_{q^t}(G)].\end{aligned} \tag{25}$$

We can similarly implement gradient descent, Nearest Newton, or Brouwer updating for iterative focusing of qp KL distance for other focusing functions besides the Boltzmann function. For example, for the logistic function as the focusing function, we get the same formulas as above, just with $G(x)$ replaced throughout by $\frac{\ln[1+e^{\beta(G(x)-K)}]}{\beta}$.

5. Summary of Update Rules

In conventional optimization over Euclidean spaces, one can use many different algorithms, including gradient descent, conjugate gradient, Newton's method, quasi-

[n]In the limit where $\tilde{q}$ is always set to the q just before the current one, our gradient descent becomes a second order dynamical equation. Iterative focusing based on several previous distributions, not just the single distribution $\tilde{q}$, leads to higher-order dynamics.

Newton, simulated annealing, genetic algorithms, etc. As proven in [35], it is theoretically impossible for any one of these problems to be superior to the others overall. Rather which algorithm one should use depends on the nature of the function being optimized, the expense of evaluating various kinds of information during the optimization, etc.

Similarly, in the preceding discussion many different PC update rules were presented, with the ultimate choice of which rule to use depending on the nature of the function being optimized, the expense of evaluating various kinds of information during the optimization, etc. This section compares some of those update rules.

All the update rules described above can be written as multiplicative updating. The following is a list of the update ratios $r_{q,i}(x_i) \triangleq q_i^{t+1}(x_i)/q_i^t(x_i)$ of some of those rules. In all of these F_G is a probability distribution over x that never increases between two x's if G does (e.g., a Boltzmann distribution in $G(x)$). In addition const is always a scalar that ensures the new distribution is properly normalized and α is a stepsize.[o]

Gradient descent of qp distance to F_G:

$$1 - \alpha[\frac{E_{q^t}(\ln[F_G] \mid x_i) + \ln(q_i^t(x_i))}{q_i^t(x_i)}] - \frac{\text{const}}{q_i^t(x_i)} \tag{26}$$

Nearest Newton descent of qp distance to F_G:

$$1 - \alpha[E_{q^t}(\ln[F_G] \mid x_i) + \ln(q_i^t(x_i))] - \text{const} \tag{27}$$

Brouwer updating for qp distance to F_G:

$$\text{const} \times \frac{e^{E_{q^t}(\ln[F_G] \mid x_i)}}{q_i^t(x_i)} \tag{28}$$

Importance sampling minimization of pq distance to $F_G(x)$:

$$\text{const} \times E_{q^t}(\frac{F_G}{q^t} \mid x_i) \tag{29}$$

Iterative focusing of $\tilde{q}$ with focusing function $F_G(x)$ using qp distance and gradient descent:

$$1 - \alpha\{\frac{E_{q^t}(\ln[F_G] \mid x_i) + \ln[\frac{q_i^t(x_i)}{\tilde{q}_i(x_i)}]}{q^t(x_i)}\} - \frac{\text{const}}{q^t(x_i)} \tag{30}$$

[o]As a practical matter, both Nearest Newton and gradient-based updating have to be modified in a particular step if their step size is large enough so that they would otherwise take one off the unit simplex. This changes the update ratio for that step. See [8].

Iterative focusing of $\tilde{q}$ with focusing function $F_G(x)$ using qp distance and Nearest Newton:

$$1 - \alpha\{E_{q^t}(\ln[F_G] \mid x_i) + \ln[\frac{q_i^t(x_i)}{\tilde{q}_i(x_i)}]\} - \text{const} \tag{31}$$

Iterative focusing of $\tilde{q}$ with focusing function $F_G(x)$ using qp distance and Brouwer updating:

$$\text{const} \times e^{E_{q^t}(\ln[F_G] \mid x_i)} \times \frac{\tilde{q}(x_i)}{q_i^t(x_i)} \tag{32}$$

Iterative focusing of $\tilde{q}$ with focusing function $F_G(x)$ using pq distance:

$$\text{const} \times E_{\tilde{q}}(F_G \mid x_i) \times \frac{\tilde{q}(x_i)}{q_i^t(x_i)} \tag{33}$$

Note that some of these update ratios are themselves proper probability distributions, e.g., the Nearest Newton update ratio.

All of these update rules are invariant under rescaling of $F_G(x)$, i.e., multiplication of $F_G(x)$ by a constant (the term const changes to compensate for the new scale). When $F_G(x)$ is an exponential (i.e., Boltzmann function) of $G(x)$, such a transformation of $F_G(x)$ is equivalent to adding a constant to $G(x)$. However for the other choices of $F_G(x)$ mentioned above (e.g., $\Theta(K - G(x))$), there is no simple correspondence between this invariance of the update rule and a change to $G(x)$. In particular, for those other F_G, rescaling of $F_G(x)$ does not correspond to adding a constant to $G(x)$. Accordingly, unlike the case for exponential $F_G(x)$, those non-exponential $F_G(x)$ do not give rise to update rules that are invariant under addition of a constant to $G(x)$. The exponential $F_G(x)$ has the other nice property that rescaling $G(x)$ is equivalent to just translating the exponent constant β, i.e., to tightening $F_G(x)$ about x with low $G(x)$. Just as other $F_G(x)$ do not have nice behavior under addition of a constant to $G(x)$, they also do not have this nice character under rescaling.

Simple modifications to non-exponential F_G allow them to share these desirable characteristics. These modifications involve dynamically replacing constants in those F_G, in particular replacing constants that might otherwise be annealed according to a pre-fixed schedule. Typically in the place of such constants one uses explicit functions of $\tilde{q}$. For example, we can replace $\Theta(K - G(x))$ with $\Theta(E_{\tilde{q}}(G) - G(x))$, or with $\Theta(\pi(\epsilon, \tilde{q}, G) - G(x))$, where $\pi(\epsilon, q, G)$ is the value of $G(x)$ in the best ϵ percentile under distribution q. For either of these replacements $F_G(x)$ is invariant under both translation and scaling of $G(x)$.

Say one has a set of multiple objective functions / constraints $\{G_i\}$ rather than just a single, unconstrained function G.[p] Some of the update rules presented above

[p]For current purposes, we can cast constraints over x as objective functions, so that the expectation

can be modified to improve all of those functions simultaneously. One does this by replacing "$F_G(x)$" throughout the update rule with some function $F_{\{G_i\}}(x)$. That function is designed so that its global maxima contain a Pareto optimal x (according to the $\{G_i\}$). In addition it cannot increase (i.e., improve) in going from one x to another if any of the $F_{G_i}(x)$ increase in that transition.[q] A simple example, applicable for update rules using pq distance, is $F_{\{G_i\}}(x) = \prod_i \Theta(K_i - G_i(x))$. In practice of course, the choice of the function $F_{\{G_i\}}(x)$ and the update rule one is modifying will have a crucial effect on how prone the algorithm is to getting caught at local critical points.

These update rules can be broadly grouped into two distinct sets based on what guarantees they have. Say we can evaluate every term in each update rule in closed form, or alternatively that our Monte Carlo estimate is exact. Then there is a q-independent target distribution, p^*, and a distance measure $\mathcal{D}$, such that each update of q in the serial Brouwer version of Eq. 28 is guaranteed not to increase $\mathcal{D}(q, p^*)$. (In this case p^* is the Boltzmann distribution over values of F_G, and $\mathcal{D}$ is qp distance.) The same is true for for adaptive importance sampling minimization of pq distance, Eq. 29. (Again p^* is the Boltzmann distribution in F_G.) For small enough step size, we also have this guarantee for gradient descent of qp distance and Nearest Newton. None of the other update rules have such guarantees.

Finally, it is worth re-emphasizing that for the update rule of adaptive importance sampling minimization of pq distance, at equilibrium each q_i is independent of the distributions q_{-i}. (It's just the marginal of the Boltzmann distribution p^β.) As mentioned previously, the same property holds for the variants of the Monte Carlo process discussed in the section on modifying the Monte Carlo process for Brouwer updating discussed above. This means one can use samples from all the previous Monte Carlo blocks with impunity. You don't have to worry that the samples of those earlier blocks were formed under a different q_{-i}, and therefore would lead to a different update from the one appropriate for the current Monte Carlo block. In addition each of these two algorithms has a single equilibrium, given by the Boltzmann distribution p^β. In this, they have no local minima problems. None of the other update rules has such a set of guarantees.

6. PC Incorporating Constraints over x

Say that we only want to minimize $G(x)$ subject to a set of equality and/or inequality constraints over X that we want to enforce exactly. This means the support of q must be restricted to $x \in X$ that meet those constraints; all other x are proscribed.

of those functions equals 0 iff the underlying distribution has its support restricted to x meeting the constraints.

[q]Such a function is only a partial ordering over x. In particular, consider a change in x which improves some G_i while all the others get worse. Often we can follow that change with another in which the improving and worsening G_i flip roles, so that in aggregate all the $G_i(x)$ have shrunk, even though $F_{\{G_i\}}(x)$ cannot have decreased.

This section discusses some schemes for finding the q that optimizes $E_q(G)$ while having a support restricted to such allowed x.

6.1. *Difficulties extending PC to X-constraints*

Some work has been done on trying to enforce X-constraints by adding to the Lagrangian a penalty term $[E_q(V)]^2$ for some appropriately chosen $V(x)$ for each X-constraint and then requiring that each such term equal 0. This for example is what was done in [18].

Now in general, restricting the support of q to the x allowed by our constraints is equivalent to a set of up to $|X| - 1$ independent, new restrictions on the values of the components of the vector q, one restriction for each component x at which those constraints mean that $q(.)$ must equal 0. This can be the case even if we have only one constraint on X, if that constraint allows only a single x.

However requiring that a set of one or more penalty terms like $[E_q(V)]^2$ all equal 0 gives us only as many new restrictions on q as there are such penalty terms. So in general the number of restrictions on q we have added in requiring the penalty terms all equal 0 is fewer than the number of new restrictions we need enforced. Accordingly, it is generically impossible to use such penalty terms to restrict q so that its support is precisely those x that are allowed by the X-constrains.

In practice, this means that a penalty term approach can only enforce the constraints by over-enforcing them, i.e., by causing a q that excludes some allowed x as well as the proscribed x. In addition, especially with product distributions, adding penalty terms $[E_q(V)]^2$ to the Lagrangian can result in local minima q whose support includes proscribed x. Accordingly, descent of the Lagrangian can get trapped at q that do not enforce our X-constraints. (In general, to enforce restrictions on the primal variable of the Lagrangian — q in our case — penalty terms should grow monotonically as one goes "further and further away" from the desired region of the primal variable. This is not the case for penalty terms like $[E_q(V)]^2$ with product distribution q.) Accordingly, as demonstrated by the results in [18], while the penalty term approach can perform quite well, its performance is not perfect.

There are other approaches to implementing X-constraints in PC that are not subject to these kinds of problems. Two of them are illustrated in the next two subsections.

6.2. *X-constrained optimization using barrier and penalty functions*

To motivate a more careful approach, as usual we encapsulate the X-constraints with a windowing function $W(x)$ that equals 1 for x that satistfy the constraints and equals 0 for all proscribed x. This translates into a set of inequality and equality constraints over q:

$$q(x) = 0 \;\forall x \text{ such that } W(x) = 0 \tag{34}$$

$$q(x) \geq 0 \; \forall x \text{ such that } W(x) = 1 \tag{35}$$

We can then apply barrier function methods (for the inequality constraints) together with Lagrange parameters (for the equality constraints) in the usual way. For illustrative purposes we choose quadratic penalty terms (that each $q(x)$ must equal 0 at a proscribed x), logarithmic barrier functions, and a product distribution q.[r] We end up with a modification of the Lagrangian above:

$$\begin{aligned}
\mathscr{L}(q, \mu, \lambda, \vec{T}) = \; & \mathrm{E}_q(G) \\
& + \; \sum_i T_i[\int dx_i \; q_i(x_i) - 1] \\
& - \; \int dx \; \mu(x) W(x) \sum_i \ln[q_i(x_i)] \\
& + \; \int dx \; \lambda(x)[1 - W(x)][\prod_i q_i(x_i)]^2 \\
= \; & \mathrm{E}_q(G) \\
& + \; \sum_i T_i[\int dx_i \; q_i(x_i) - 1] \\
& - \; \sum_i \int dx \; \mu(x) W(x) \ln(q_i(x_i)) \\
& + \; \mathrm{E}_q(q\lambda[1 - W]) \\
= \; & \mathrm{E}_q(G) \\
& + \; \sum_i T_i[\int dx_i \; q_i(x_i) - 1] \\
& - \; V \sum_i \mathrm{E}_{\hat{\mu}}(W \ln(q_i)) \\
& + \; \mathrm{E}_q(q\lambda[1 - W])
\end{aligned} \tag{36}$$

where $\lambda(x)$ is a set of Lagrange/penalty parameters, the T_i are still Lagrange terms; $\mu(x)$ is a set of barrier parameters; and $V \triangleq \int dx \; \mu(x)$ is the normalization constant to turn μ into the probability distribution $\hat{\mu}$. Note that we no longer explicitly have a barrier term for keeping $q_i(x_i)$ non-negative $\forall x_i$; that's taken care of with the combination of our barrier terms and our $\lambda(x)$-based penalty terms.

This Lagrangian gives the following update rules:

$$\frac{\partial \mathscr{L}}{\partial q_i(x_i)} = E_q(G \mid x_i) + T_i - V \frac{\hat{\mu}(x_i) \mathrm{E}_{\hat{\mu}}(W \mid x_i)}{q_i(x_i)} + 2\mathrm{E}_q(q\lambda[1 - W] \mid x_i) \tag{37}$$

is the gradient, with T_i found by integrating the other terms over all x_i in the usual way. So parallel Brouwer updating is based on the set of equations

$$q_i(x_i) = V \frac{\hat{\mu}(x_i) \mathrm{E}_{\hat{\mu}}(W \mid x_i)}{T_i + 2\mathrm{E}_q(q\lambda[1 - W] \mid x_i) + \mathrm{E}_q(G \mid x_i)} \tag{38}$$

[r]These penalty terms should not be confused with the ones on expectations over q discussed above.

and Nearest Newton is based on the gradients

$$\frac{\partial \mathscr{L}}{\partial p(x)} = G(x) + T - \frac{\mu(x)W(x)}{p(x)} + \lambda(x)[1 - W(x)]p(x) \tag{39}$$

and (diagonal, positive-definite) Hessians

$$H_{p(x),p(x)} = \frac{\mu(x)W(x)}{p^2(x)} + \lambda(x)[1 - W(x)]. \tag{40}$$

6.3. *Monte Carlo implementations of enforcing X-constraints via barrier and penalty functions*

Regardless of the barrier functions and penalty terms we use, the conditional expectations in this update rule can be evaluated via Monte Carlo, as usual. The difference is that, in addition to computing $G(x)$ for each joint move x in the Monte Carlo process, we also need to compute $W(x)$. Note that the actual “computation” of the objective function $G(x)$ and the feasibility indicator $W(x)$ has nothing to do with the optimization algorithm. These computations are performed by an oracle, which represents the process or entity we are trying to optimize: it could be a complex computer program that computes performance measures and feasibility of some design, or an actual physical process whose performance is to be optimized, the response of players in a game, or even just nature’s responses to some actions by many “agents”.

An interesting issue that now arises though is how we evaluate the parameter functions $\mu(x)$ and $\lambda(x)$ which (just like q) get updated periodically, and which also need to be broadcast at every Monte Carlo step in order to for every agent i to calculate the update to q_i. This is particularly crucial for the Lagrange parameter vector $\lambda(.)$; we must eventually get it exactly right to be guaranteed that the associated (penalty term) constraints on q are satisfied. (The barrier functions constraints on q are enforced no what matter barrier parameter vector $\mu(.)$ we use, so long as none of its components exactly equals 0.) There are several ways to implement this dynamics of $\lambda(.)$. Here we describe one where the same oracle that evaluates $W(x)$ with each Monte Carlo sample also evaluates $\lambda(x)$.

In first order ascent schemes for finding critical points of the Lagrangian, the Lagrange parameters are updated by gradient ascent in the Lagrangian each time one finds a minimum over the primal variable. Here that means each $\lambda(x)$ is held fixed while one minimizes the Lagrangian over q. Then each $\lambda(x)$ gets incremented by (a stepsize times)

$$\frac{\partial \mathscr{L}}{\partial \lambda(x)} = [1 - W(x)]q(x) \tag{41}$$

once we have found such a local minimum (over q) of the Lagrangian. Then we again minimize the Lagrangian over q, for this new parameter vector, and repeat the process.

On the face of it, this would seem to require that we store values of the function $\lambda(x)$ for all proscribed x. (Note we never update $\lambda(x)$ for allowed x, since the right-hand side of Eq. 41 always equals 0 for any such x.) However we only need to use those values when we estimate conditional expectations over q using Monte Carlo, i.e., when we have a particular joint move x. This allows us to do the update to the $\lambda(x)$ associated with that Monte Carlo sample x on the fly.

The basic idea is to note that for a proscribed x, the cumulative effect of all previous updates to $\lambda(x)$ is the sum of the associated preceding values of $q(x)$ (since $1 - W(x) = 1$ always for such an x). So say we have a Monte Carlo joint move x. Have each agent i broadcast the vector of the probabilities it assigned to x_i at all preceding times the system was at a (q-space) minima of the Lagrangian, $\{q_i^{t'} : t' \text{ a preceding minimum of } \mathscr{L}\}$. The oracle that is calculating whether the joint Monte Carlo move x is feasible (i.e., checking if $W(x) = 0$) receives all those vectors. If x is proscribed, the oracle calculates the "dot product" $\sum_{t'} \prod_i q_i^t(x_i)$. This is exactly the current value of $\lambda(x)$ (assuming $\lambda(x)$ started at 0). The oracle can then broadcast that value out to all the agents. Each agent i then uses the set of $\lambda(x)$ values it has received from the oracle those times it made Monte Carlo move x_i to estimate $E_q(\lambda[1 - W] \mid x_i)$. (It does this exactly the same way it uses the broadcast $G(x)$ values to estimate $E_q(G \mid x_i)$.)

6.4. *Small regions of allowed X when using barrier and penalty functions to impose X-constraints*

The foregoing will run into difficulties if the constraints on x are so tight that the feasible region is almost never sampled. In the extreme case, if we're doing a satisfiability problem, finding a feasible x is the whole goal, and encountering one in random sampling will be rare, to put it mildly.

For such problems I think one has to deal with the equality constraints using penalty functions rather than Lagrange parameters. Formally, a penalty function is a function that equals 0 if the constraints are satisfied, and is greater than 0 otherwise. Such functions are added in to the objective function. (If they're used for equality constraints, then they replace the usual Lagrange terms enforcing those equality constraints.) This means that sample points outside of the feasible region will now contribute something other than 0 to the associated expectation values. Augmented Lagrangians are an example of all this.

Typically one uses penalty functions to deal with all the constraints of the problem, both equality and inequality. For simplicity, I'll consider a modification, where one is still using entropy barrier functions to deal with the inequality constraints, and Lagrange parameters to ensure each q_i is normalized, but use penalty functions to deal with the equality constraints specifying the feasible region.

A typical case is the quadratic penalty function. The resultant Lagrangian is

$$\mathscr{L}(q, \mu, \nu, \vec{T}) = \mathrm{E}_q(G)$$

$$
\begin{aligned}
&+ \sum_i T_i[\int dx_i \; q_i(x_i) - 1] \\
&+ V \sum_i \mathrm{E}_{\hat{\mu}}(W \; q_i \ln(q_i)) \\
&+ \mathrm{E}_q(\nu[1 - W]q). \qquad (42)
\end{aligned}
$$

The associated gradient is

$$
\begin{aligned}
\frac{\partial \mathscr{L}}{\partial q_i(x_i)} = &\; E_q(G \mid x_i) + T_i \\
&+ V\hat{\mu}(x_i)\mathrm{E}_{\hat{\mu}}(W \mid x_i)[1 + \ln(q_i(x_i))] \\
&+ 2\mathrm{E}_q(\lambda[1 - W]q \mid x_i) \qquad (43)
\end{aligned}
$$

and similarly for the other update rules.

6.5. *X-constrained optimization using pq KL distance*

The foregoing can be viewed as modifying the qp KL distance and associated descent schemes to enforce our constraints. This raises the obvious idea of trying to enforce such constraints by appropriately modifying pq KL distance and the associated adaptive importance sampling technique.

One way to do that is to simply multiply the target distribution $p(x) = F_G(x)$ by the windowing function $W(x)$ discussed above (e.g., multiply the Boltzmann distribution, $F_G(x) = p^\beta(x)$, by $W(x)$). We then conduct adaptive importance sampling as usual, only using that product in place of F_G throughout. The usual guarantees go through. For example, if there is a single (constrainted) global minimum of G, x', and if the importance sample estimates of the marginals are exact, then that constrained minimum will be found in the limit that $F_G(x) \to \delta(x - x')$ (e.g., in the limit that $\beta \to \infty$).

Note that such windowing of F_G by W is usually of no help for qp KL distance. That's because after that multiplication the term $E(G \mid x_i)$ arising in the update rules based on qp distance gets replaced by $E_q(G + \ln[W] \mid x_i)$. This is undefined, i.e., infinite, if you do not have a q whose support is restricted to the allowed x. Yet typically, that is precisely the situation early in the optimization process.

7. Uncountable X

There are several circumstances in which naive empirical averaging of Monte Carlo samples to estimate update terms of the form $E(F_G \mid x_i)$ will not work. For example, consider the simplest situation, in which we have a finite number of agents and a finite move space for each agent. Even in this situation, if there are not enough Monte Carlo samples, it may be that for some potential move of some agent there are no instances in any of the Monte Carlo samples (in any of the blocks) in which that agent made that move. In that case, we cannot use empirical averaging to estimate the associated $E(F_G \mid x_i)$. As another example, say we have a large (but finite) number of Monte Carlo samples, but some agent has an uncountable number of potential moves. Then that agent will have no samples for almost all of its potential moves.

7.1. *Exploiting supervised learning*

All of these problems can be addressed by exploiting the fact that we are working with a product distribution, in concert with the techniques from the field of supervised learning techniques (i.e., classification and regression) [31], which concern precisely the issue of estimating $E(F_G \mid x_i)$ from a finite set of Monte Carlo samples. As an example, consider the first problem case mentioned above, in which there a finite number of agents all with a finite number of potential moves, but we have too small a set of Monte Carlo samples to have samples of all moves for all agents. For this scenario each agent i must estimate $E(F_G \mid x_i)$ for all x_i using a "training set" of Monte-Carlo-generated (x_i, F_G) pairs that does not extend over all x_i. This is a standard problem in supervised learning [31]. Often it can be addressed by extrapolating from those x_i which did occur in the training set to infer estimates of $E(F_G \mid x_i)$ for the x_i that did not. Those estimates can then be used to form the updates for those non-arising x_i. The simplest version of such a scheme is to set $E(F_G \mid x_i)$ for an unsampled x_i to the average of the F_G values in the training set.[s] However often more sophisticated schemes can be used, based upon prior knowledge concerning the likely dependence of $E(F_G \mid x_i)$ on x_i.[t]

Similar techniques can be used even when the x_i are uncountable. Moreover, in general a supervised learning fit to the Monte Carlo data is parameterized by a finite set of numbers, and therefore for a finite number of agents those fits can be stored in a finite computer, regardless of the cardinality of the move spaces of the agents. However for uncountable move spaces we have the extra problem of how to store, update, and sample q, which is now a density function rather than a probability distribution.

Fortunately, given the regression $E(F_G \mid x_i)$, there are several ways to update and sample $q(x)$ without ever explicitly storing the values of $q(x)$ for all possible x. By using such sampling schemes in concert with the regression scheme, we can implement Monte Carlo updating for all of the problematic scenarios described above. As outlined in this section, the key is to write the update rules in terms of multiplicative update ratios giving the new q in terms of the old one, as in the list presented above.

[s]In gradient descent updating this means that $q_i(x_i)$ for an unsampled x_i does not change at the update step.

[t]In such scenarios the data in the training set should not only be used to form estimates of $E(F_G \mid x_i)$ for those x_i that don't occur in the training set; it should also be used to refine our estimates for those x_i values that *do* occur in the training set.

7.2. *Uncountable x and finite parameterizations of q*

For all of these update rules listed above, when x_i is a compact subset of a Euclidean space, one can still numerically perform the update in the conventional way if the associated probability density function is replaced by a (finite-dimensional) parameterization of it. The simplest way to do that is, in essence, by binning x_i. This means that agent i now has a finite set of moves, one for each of its bins. The full density function is parameterized by the real numbers giving the probabilities agent i assigns to each of its bins, according to some pre-set rule. One example is where the probability density function has uniform density in each bin (as in Reimann integration). Another is where the density function is linearly increasing/decreasing across each bin, in such a way that the density function is everywhere continuous (as in the trapezoidal rule for integration). Formally, such binning schemes are semi-coordinate transformations [9, 11].

With such a scheme, one first applies supervised learning techniques to the Monte Carlo samples to determine the regression $E(F_G \mid x_i)$. For each bin j, having borders a_j and b_j, one then numerically computes two integrals:

$$\int_{a_j}^{b_j} dx_i \; q_i^t(x_i) E(F_G \mid x_i) \text{ and } \int_{a_j}^{b_j} dx_i \; q_i^t(x_i).$$

The ratio of those two integrals determines the time-t expected value of F_G conditioned on x_i being in bin j. (For bins that are thin enough on the scale of variations in the regression and/or $q_i^t(x_i)$, these integrations can be replaced by simply evaluating the integrands at the centers of the bins.) This then gives the expected F_G conditioned on x_i being in bin j for all bins j. This is precisely what is needed to update those bins' probabilities, according to whichever of the update rules listed above one is using.

Note that this scheme can be done even when the number of bins is far larger than the number of Monte Carlo samples. This contrasts with the case of estimating the conditional expectation value of F_G given bin j based only on averaging of all the Monte Carlo samples that fall in that bin. Intuitively, using regression allows samples from neighboring bins to be used to help form the estimate.

While some binning schemes can be relatively sophisticated [9], sometimes it would be advantageous to use a different parameterization. Often this can be done in a way that replaces the regression algorithm with a density estimation algorithm, using the usual Bayesian equivalence of regression and density estimation. For example, choose the masking function $F_G(x)$ in Eq. 33 to be $\Theta(K - G(x))$, Evaluating such an update based on a set of Monte Carlo samples can be done with conventional probability density estimation algorithms [31]. One simply collects the subset of the samples for which $G(x) < K$, and runs the density estimation algorithm on those points to estimate the density at x_i.

Intuitively, in this approach the Monte Carlo samples encode the probability density function q_i. For a smooth density estimator, this scheme will also ensure $q_i(x_i) \neq 0 \; \forall x_i$, thereby mitigating the problem that a statistical fluctuation of never picking x_i in some Monte Carlo block would guarantee it is never picked in the future. Similar schemes can be used for non-step function choices of F_G. For example, one can use the value $F_G(x)$ for each x in the Monte Carlo sample as a weighting factor for that sample in a kernel density estimator.

7.3. *Parameterless sampling via Sample Correction*

One problem with parametric schemes like these is that since q is given by a set of explicitly stored real numbers, one is always limited in how finely one can capture q by the finiteness of one's computer's memory. More importantly, if one has many parameters (to capture q with high accuracy), then updates can be computationally expensive, since each parameter has to be updated in each iteration. For example, with binning, one has to go through the update rule for each bin.

Alternative schemes use the regression $E(F_G \mid x_i)$ to apply any multiplicative update rule for uncountable x without any finite-dimensional parameterization of q. With such schemes, in each step the full density function given by an uncountable number of real numbers is implicitly updated (e.g., via gradient descent). However that density is never explicitly represented. Instead, all we ever explicitly do is sample it, potentially evaluating it at a finite number of points to do so. Intuitively, via our regression, the Monte Carlo samples themselves serve as our "parameterization" of $q(x)$.

Define $R_{q,i} \equiv \max_{x_i} r_{q,i}(x_i)$. Then for any $t > 1$ we can generate a sample from q_i^t if we can implement the following three-step sample-correcting procedure based on subsampling:

1) Sample from q_i^{t-1} to get a point x_i.

2) Toss a coin with probability of heads

$$\frac{r_{q^{t-1},i}(x_i)}{R_{q^{t-1},i}}. \tag{44}$$

(The reason for dividing by $R_{q^{t-1},i}$ is to ensure this probability of heads never exceeds 1.)

3) If the coin came up heads, keep our x_i as the desired sample of q_i^t. Otherwise return to (1).[u]

This scheme is an instance of **sample correction**, which is discussed in a more general context in the next section. Note that this particular type of sample cor-

[u]This is essentially importance sampling. Formally, since this three-step sub-sampling scheme is a stochastic process, it generates x_i's according to some distribution $\pi(x_i)$. So to prove that $\pi = q_i^t$, it suffices to note that for any two values x_i, x_i', $\frac{\pi(x_i)}{\pi(x_i')} = \frac{q_i^t(x_i)}{q_i^t(x_i')}$. (This is proven formally in the appendix.) **QED**

rection will also work if we can only evaluate the values $r_{q^{t-1},i}(x_i)$ up to an overall proportionality constant, so long as $R_{q^{t-1},i}$ is redefined to include that constant. Similarly the scheme will work so long as $R_{q^{t-1},i}$ in step (2) is replaced by any fixed quantity that is bounded below by the actual $R_{q^{t-1},i}$. So in practice we can set that value in step (2) to some small factor greater than 1 multiplied by the maximal value of over some set of values x'_i of $r_{q^{t-1},i}(x'_i)$. Accordingly we can sample q^t if we can sample q^{t-1}, can evaluate $A \times r_{q^{t-1},i}(x_i)$ for any particular x_i and fixed (though perhaps unknown) constant A, and can evaluate an upper bound on $A \times R_{q^{t-1},i}$.

Performing our subsampling procedure for all agents will give us a sample of the joint distribution q^t. We then add that joint sample to the training set and form a new regression (to be able to calculate $r_{q^t,i}(x_i)$). If we need to do so to ensure the quantity in step (2) never exceeds 1, we then use that new regression to find an upper bound on $R_{q^t,i}$. This allows us to repeat the three steps, and thereby form the next update to q. Generalizing, if we set q^1 to some easily sampled distribution (e.g., the uniform distribution), and can always perform the stipulated regressions, then with our subsampling procedure we have an iterative algorithm for sampling $q^t(x) = \prod_i q_i^t(x)$ for all t. Then, at the end of the run, we use the final joint samples as guesses for the solution x to our optimization problem.

Say we are at iteration t, having formed samples of all of the $q_i^{t'}$ for $t' < t$ via the subsampling procedure, and therefore having been able to evaluate $R_{q^{t'},i}$ and $r_{q^{t'},i}(x_i)$ for any $x_i, t' < t$. To employ the precise scheme outlined above to sample q_i^t we would first sample q_i^1, and then send that sample through t successive stochastic keep/reject steps. The probability of a rejection at each step in that chain is given by how small the ratio $\frac{r_{q^{t-1},i}(x_i)}{R_{q^{t-1},i}}$ is for typical x_i generated by sampling q_i^1. For large enough t, even if the rejection probability for each step in the chain is small, the probability of a rejection somewhere along such a chain — followed by starting all over with a new sample of q_i^1 — may be quite high. Accordingly, this subsampling procedure might take a long time to actually generate the desired sample of q_i^t.

As an alternative, note that by hypothesis we can evaluate $r_{q^{t'},i}(x_i)\ \forall x_i,\ t' < t$, up to a t'-dependent overall proportionality constant, which without loss of generality we set to 1. So write

$$q_i^t(x_i) = q_i^1(x_i) \prod_{t'=1}^{t-1} r_{q^{t'},i}(x_i)$$

As long as we are sure that the product on the righthand side is finite and never negative, we can employ a modified version of our sub-sampling procedure. To do this define

$$c_i(\{q^{t'} : t' < t\}, x_i) \equiv \prod_{t'=1}^{t-1} r_{q^{t'},i}(x_i). \qquad (45)$$

Assuming we can evaluate $r_{q^{t'},i}(x_i)\ \forall, x_i, t' < t$, we can evaluate $c_i(\{q^{t'} : t' <$

$t\}, x_i)\ \forall x_i$. Next define

$$C_i(\{q^{t'} : t' < t\}) \equiv \max_{x_i} c_i(\{q^{t'} : t' < t\}, x_i). \tag{46}$$

In analogy to the earlier case, we can form an estimate of a (conservative lower bound) on $C_i(\{q^{t'} : t' < t\})$ by evaluating $c_i(\{q^{t'} : t' < t\}, x_i)$ for many x_i.

As before, the first step of our procedure is to sample q_i^1 to produce a suggested sample of q_i^t. We then accept that suggested sample with probability

$$\frac{c_i(\{q^{t'} : t' < t\}, x_i)}{C_i(\{q^{t'} : t' < t\})},$$

resampling q_i^1 if we reject the suggested sample. This gives us our desired sample of $q_i^t(x_i)$. Doing this for all i then gives our sample of $q^t(x)$.

Exactly as before, such a sample of q^t can be combined with our previous Monte Carlo samples to provide a training set for a supervised learning algorithm that forms a regression $E_{q^t}(F_G \mid x_i)$. We can use that to evaluate $r_{q^t,i}(x_i)$ for any x_i, up to an overall proportionality constant. So we can evaluate the product $c_i(\{q^{t'} : t' < t+1\}, x_i)$ for a large number of x_i, and thereby estimate (an upper bound on) $C_i(\{q^{t'} : t' < t\})$. This then allows us to generate a sample of the next distribution q^{t+1} by using subsampling. So we again have an iterative algorithm. However this way one avoids the need for more than one keep/reject step in forming the sample of q^t for any t. (The price paid for this is a more expensive numerical evaluation of the associated max.)

7.4. *Including density estimation*

A remaining potential difficulty is that as q_i^t gets more and more peaked, we might get a lot of rejections when we subsample, since the ratio $\frac{c_i(\{q^{t'}:t'<t\},x_i)}{C_i(\{q^{t'}:t'<t\})}$ will be very small for almost every point formed by sampling q_i^1. More generally, if we are only generating candidate x_i by examining points generated by sampling q_i^1, then we won't have reduced the overall computational burden in finding x with low G values compared to the simple process of sampling q_i^1 without any subsequent subsampling.

We can address this problem by periodically using a density estimation algorithm to produce an estimate of the current distribution, an estimate that is easy to sample. However we don't directly use that estimate in our algorithm in place of q_i^t, since it won't exactly equal q_i^t in general. Instead, we use it as a proposal distribution in importance sampling from q_i^t. In essence, we use the same keep-reject procedure as before, only with a non-uniform distribution generating samples, and doing so after q has already started evolving.

More precisely, at time step t, say we run a density estimation algorithm on our Monte Carlo samples to form a density $\hat{q}_i^t(x_i)$ that both can be easily sampled and with high probability is a good approximation to q_i^t. Write

$$q_i^{t''}(x_i) \propto \hat{q}_i^t(x_i)\, d_i(\{q^{t'} : t' < t''\}, \hat{q}_i^t, x_i) \tag{47}$$

where

$$d_i(\{q^{t'} : t' < t''\}, \hat{q}_i^t, x_i) \equiv \frac{q_i^1(x_i) c_i(\{q^{t'} : t' < t''\}, x_i)}{\hat{q}_i^t(x_i)}. \tag{48}$$

Then define

$$D_i(\{q^{t'} : t' < t''\}, \hat{q}_i^t, i) \equiv \max_{x_i} d_i(\{q^{t'} : t' < t''\}, \hat{q}_i^t, x_i). \tag{49}$$

As usual, without loss of generality we can ignore any overall proportionality constants in the evaluations of $\hat{q}_i^t(x_i)$ and/or $c_i(\{q^{t'} : t' < t''\}, x_i)$ (so long as the same constants appear in the evaluation of $D_i(\{q^{t'} : t' < t''\}, \hat{q}_i^t, i)$), and can replace the constant $D_i(\{q^{t'} : t' < t''\}, \hat{q}_i^t, i)$ with an upper bound on it.

In the first step of the new version of our subsampling procedure — when we want to generate a sample of $q_i^{t+1}(x_i)$ — we start by generating a sample of $\hat{q}_i^t(x_i)$. (In the original subsampling procedure the analogous step was to sample $q^1(x_i)$.) We then keep that sample with probability

$$\frac{d_i(\{q^{t'} : t' < t+1\}, \hat{q}_i^t, x_i)}{D_i(\{q^{t'} : t' < t+1\}, \hat{q}_i^t, i)},$$

forming a new sample if the suggested sample is rejected. In this way we can exactly sample the density function $q_i^t(x_i)$. Moreover, assuming our density estimate is reasonably accurate, and that our upper bound on $D_i(\{q^{t'} : t' < t+1\}, \hat{q}_i^t, i)$ is not too much greater than the actual value, the ratio giving our acceptance frequency will not be too small. We then proceed analogously for times $t'' > t+1$.

In practice, we may want to exploit algorithms that combine the generation of $\hat{q}_i^t$ from the training set and the sampling of that distribution. As an illustration, say x_i is the set of real numbers between 0.0 and 1.0, and write the cumulative distribution function of q_i^t as $CDF_{q_i^t}$. Then one way to form a sample of $q_i^t(x_i)$ is to generate a point ξ_i by uniformly sampling [0.0, 1.0], and then return the value $[CDF_{q_i^t}]^{-1}(\xi_i)$. This suggests an algorithm in which we first use our training set of Monte Carlo samples to form $\widehat{CDF}_{q_i^t}$, an estimate of $CDF_{q_i^t}$. We then sample [0.0, 1.0] uniformly to produce ξ_i, and return the value $[C\hat{D}F_{q_i^t}]^{-1}(\xi_i)$.

As an example of a rough, fast way to do this, say there are N separate x_i values in our training set, the set of those values being written as $\{x_i^j\}$. Define $I(x_i^j)$ as the interval of all real numbers that are closer to x_i^j than to any other training set element. Also define the function $\text{int}(x_i)$ as the greatest integer below x_i. So if ξ_i is a real number chosen by randomly sampling (0, 1.0), $\text{int}(N\xi_i) + 1$ is a uniformly random choice of one of the N elements of the training set. Using this, our algorithm for sampling (an estimate of) $q_i^t(x_i)$ would consist of the following steps:

A) Sample [0.0, 1.0] uniformly to generate ξ_i, and then set $j \equiv \text{int}(N\xi_i) + 1$.

B) Sample uniformly from within the interval $I(x_i^j)$.

Intuitively, under this scheme the density estimate we sample is uniform within each interval $I(x_i^j)$ (one such interval for each j). The value of the estimate in each interval $I(x_i^j)$ (one such interval for each j) is proportional to the inverse of the width of the interval, $|I(x_i^j)|$, with the same proportionality constant for all intervals. So where training set elements are dense, interval widths are small, and density estimates are large. Similar schemes can be used when x_i is more than one-dimensional, for example by substituting Voronoi cells about training set x_i values for intervals about them.

Note that we actually have far more information about the underlying density to use in forming our estimate than just the x_i values in the data set: we have the associated values of the actual density at those points, $\{q_i(x_i)\}$. Indeed, one could imagine forming our density estimate without using a conventional unsupervised learning density estimator at all, but instead a supervised learning regression scheme. Such a scheme would form an $x_i \rightarrow q_i(x_i)$ "fit" to the $\{(x_i, q_i(x_i)\}$ pairs comprising our data set, constraining the fit to be nowhere negative and integrate to 1. However using regression this way also doesn't fully exploit our information: it ignores the fact that the positions $\{x_i\}$ were formed by sampling q_i.

A proper Bayesian approach would exploit both sets of information. However there are alternative, less formal ways to exploit both sets of information. An example is a modification of the (A-B) algorithm presented just above. In this modification we still have our density estimate be constant within each interval $I(x_i^j)$, but change the value of that estimate to incorporate the (known) density value at x_i^j, $q_i(x_i^j)$. Then rather than have the density within each interval $I(x_i^j)$ proportional to $1/|I(x_i^j)|$, we set it to be proportional to $\sqrt{q_i(x_i^j)/|I(x_i^j)|}$. (The square root is motivated by considering what happens if $q_i(x_i)$ is multiplied by some constant k across a certain region, which would mean that the training set elements will, on average, be spaced k times more densely in that region.) A similar modification would instead replace $1/|I(x_i^j)|$ with $\frac{1/|I(x_i^j)|+q_i(x_i^j)}{2}$. Note that in either modification we must solve for the proportionality constant, unlike in the original scheme. This is straight-forward however.

Say we are able to sample $q_i^t(x_i)$ exactly, either by using subsampling of points generated from q_i^1 or by using a density estimate $\hat{q}_i^t$. Then we can use the original subsampling procedure recounted above to sample $q_i^{t+1}(x_i) = q_i^t(x_i) r_{q^t,i}(x)$. Similarly, to sample $q_i^T(x_i)$ for subsequent $T > t+1$, we can use the modified version of the subsampling procedure based on a product of $r_{q^{t'},i}(x_i)$'s. In the current context, this means we sample $q_i^t(x_i)$, and then keep/reject those samples according to the ratios

$$\frac{c_i(\{q^{t'} : t < t' < T\}, x_i)}{C_i(\{q^{t'} : t < t' < T\})}.$$

Once T is so much larger than t that we start getting a lot of rejections, we can rerun our density estimation algorithm.

Finally, while we lose formal bounds in doing so, we may elect not to use c_i's that reflect products of r_i's all the way from time 1. Say we make a density estimate at some time, and it is a particularly accurate one. Then we may want to start the entire subsampling procedure afresh, *using that density estimate rather than the uniform distribution as our initial distribution.* This would mean that each c_i only reflects the product of r_i's for the times since that most recent density estimate. In essence, in this variant, all of our work up to the formation of that most recent density estimate was simply a procedure for finding a starting distribution for the subsampling procedure, a starting distribution that (hopefully) has a low value of our Lagrangian.

7.5. *Performing the needed evaluations*

Say we are at the t'th iteration, and assume we already have a full Monte Carlo sample of q^{t-1}. We need to be able to evaluate $r_{q^{t-1},i}(x_i)$ to form a sample of q^t using our subsampling procedure. We can do this for any of the update rules listed above, so long as can calculate $\ln(q_i^{t-1}(x_i))$, $E_{q^{t-1}}(\ln(F_G) \mid x_i)$, $E_{q^{t-1}}(\ln(F_G))$, $S(q_i^{t-1})$, $\int dx_i \; E_{q^{t-1}}(\ln(F_G) \mid x_i)$, and $\int dx_i \; \ln(q_i^{t-1}(x_i))$. The first two of these depend on x_i, and the last four are averages over all x_i.[v]

All of these terms have to be calculated to update q even if one doesn't use subsampling. In particular this is the case if one converts a scenario of infinite x's into one where each agent has a finite number of moves by dividing the range of each x_i into a large number of bins, and then uses conventional (non-subsampling) PC. The difference is that with subsampling we cannot just look up $q_i(x_i)$ values.[w]

We can perform our needed evaluations as follows:

i) We can evaluate $q_i^{t-1}(x_i) \; \forall x_i$ by direct expansion. It is a product of the values of $r_{q^{t'},i}(x_i)$ for $t' < t$ with the value of a starting density at x_i, and by the inductive hypothesis we can evaluate all of those values. That takes care of the first term.

ii) As usual, to estimate $E_{q^{t-1}}(\ln(F_G) \mid x_i)$, apply any handy supervised learning algorithm (e.g., Gaussian nearest neighbor averaging) to the training set of $(x_i, \ln(F_G(x)))$ pairs given by the Monte Carlo samples of q^{t-1}.

iii) Given our supervised learning algorithm, use numerical integration to estimate $\int dx_i \; E_{q^{t-1}}(\ln(F_G) \mid x_i)$. In practice, if the integrand is quite peaked, it may make sense to assist the integration by first using a density estimation algorithm to form an easily sampled estimate of $q_i^t(x_i)$. Numerical integration via importance sampling

[v]Those last four arise in calculating the additive const term in one or the other of the update rules, and where needed implicitly assume *a priori* bounds on their integrals. Note that any multiplicative const terms are irrelevant, since as described above they cancel out in the subsampling.

[w]An additional difference is that with subsampling we may need to do periodic density estimation, to keep the rejection frequency from growing too large. But that's not necessary just to perform a particular update at the end of a Monte Carlo block.

can then be used with that estimate as the proposal distribution to do the integration. Assuming the peaks of $q_i^t(x_i)$ roughly match those of $E_{q^{t-1}}(\ln(F_G) \mid x_i)$, such integration should be relatively efficient.

Given our assumed ability to evaluate $q_i^{t-1}(x_i)\ \forall x_i$, we can similarly use our supervised learning algorithm and numerical integration to estimate $E_{q^{t-1}}(\ln(F_G))$, again using density estimation if need be. Alternatively, we can estimate $E_{q^{t-1}}(\ln(F_G))$ simply by averaging the values in the training set of $\ln(F_G)$.[x]

iv) Similarly, we can use numerical integration to estimate $\int dx_i \ \ln(q_i^{t-1}(x_i))$ and/or $S(q_i^{t-1})$. A simpler approach to estimating the entropy, analogous to the averaging process of step (iii), is to simply estimate the entropy as the empirical average of the values of $\ln[q_i(x_i)]$ over the Monte Carlo samples. Similarly, an importance-sample estimation of $\int dx_i \ \ln(q_i^{t-1}(x_i))$ would be given by the empirical average of the values of $(\ln[q_i(x_i)])/q_i(x_i)$.

Doing all this, we can evaluate every term that arises in our subsampling procedure. This allows us to sample q^t. For the next iteration, this ability to sample q^t is used again, this time to do part (1) of the subsampling procedure for generating samples of q^{t+1}. To perform parts (2) and (3) we need to evaluate the update ratio, and therefore must be able to perform (some subset of) steps (i) through (iv) above. Since by hypothesis we can evaluate $q^{t-1}(x)$ for any x, and can evaluate the update ratio $r_{q^{t-1},i}(x_i)$ (up to irrelevant proportionality constants) for any such x_i, we can evaluate $q^t(x)$ for any x. Therefore we can perform step (i). We can also perform steps (ii) through (iv) using the Monte Carlo samples of q^t. Therefore we can perform parts (2) and (3) of our subsampling procedure. So we can generate a sample of q^{t+1}.

7.6. *Bootstrapping to use all the data and practical issues*

In practice we will often not perform the updating exactly as specified above. It may be unwieldy to work with distributions whose updating history all the way back to the uniform distribution is maintained. In such a situation, if we think that the density estimate is a reasonably accurate approximation of q^t, then it may make sense to "restart the entire process" with that estimate replacing the uniform distribution. In other words, we would set t back to 0, but rather than use the uniform distribution as our initial distribution, we would use our current density estimate. In general we would expect that updating distributions starting from that density estimate should outperform updating distributions from a uniform distribution. On the other hand, updating from q^t should perform even better still. So clearly we would not want to perform this kind of restarting from the density estimate too frequently.

[x]It probably makes most sense to do this if our supervised learning algorithm is one for which we're *a priori* guaranteed that such averaging gives the same answer as numerical integration.

A more important modification of the subsampling procedure addresses the fact that as recounted above, it throws out many of the Monte Carlo samples. This clearly is wasteful. There is also an intuitive problem with it; if the random keep/reject steps had come out differently, then a different subset of the Monte Carlo samples would be kept. This suggests that there is nothing special about one such subset rather than another one.

The most straightforward way to address these concerns is to bootstrap the keep/reject step. In this modification one makes many passes through the Monte Carlo samples, each time forming a separate set of (random) keep/reject decisions, and forming a separate estimate of how to update the distributions. Our final estimate is then given by averaging those separate updates.

In some situations we can actually go to the limit of an infinite number of passes in closed form. As an example, consider the case where X contains a finite number of points. Say that the density estimator for a single bootstrap pass for each agent i works by forming the frequency counts with which the values x_i arise in the subset of the Monte Carlo points kept in that pass. Then because frequencies converge to averages, averaging the density estimate of an infinite number of passes will produce an estimate where $q_i(x_i)$ is set to the (average over the samples) keep/reject ratio associated with move x_i in the Monte Carlo samples. Intuitively, that (average) ratio serves as a weight in the density estimate.

8. Beyond Product Distributions: Sample Correction

An important subsampling scenario is where one uses a single agent. In this situation product distributions may still arise — in the density estimation algorithm. Alternatively, other graphical models besides product distributions can be used.

However many agents it is used with, subsampling is an example of a general class of schemes that replace the usual update rules with **sample-corrected** versions. This term refers to algorithms for forming a sample of a specified distribution that is hard to sample directly (e.g., the distribution given by an application of an update rule). These algorithms start by forming an IID sample of some different proposal distribution. They then stochastically "correct" that sample to get a sample of the specified distribution. That corrected sample may then be used to update the proposal distribution, though this isn't always necessary.

This section first discusses other schemes for sample correction besides subsampling, as well as how sample correction can be used even when has only a single agent. It then discusses single agent subsampling, and many of its advantages, e.g., how it can be used to form IID samples of an arbitrary provided distribution.

8.1. *Sample correction without subsampling and general comments*

As mentioned above, there are numerous schemes besides subsampling that do sample correction. As an example, say we are given a desired distribution $p(x)$ and a proposal distribution $\hat{p}(x, x')$. Then the Metropolis-Hastings algorithm [13] is a way to use $\hat{p}$ to produce a random walk of which, in the ergodic limit, is an IID sample of p. To use this algorithm one only needs to be able to sample $\hat{p}(., x')$ for any x' and to evaluate $p(x)$ at any x. In particular one does not need to explicitly store p for all x. Just as with subsampling then, we can use the Metropolis Hastings algorithm with any of our update rules to generate a sample of the updated distribution, so long as we can evaluate all the terms in the update equation. This allows us to produce the desired sample of the updated distribution.

There can be drawbacks to such alternatives however. In particular, the theory underpinning the Metropolis-Hastings algorithm assumes the proposal distribution never changes in time, an issue that can be addressed only with some difficulty [13]. In addition, subsampling has some advantages absent from these alternative schemes (e.g., the use of the constant k to determine the degree of sample correction, discussed below).

However it is done, sample-correcting the update rules may be helpful even when the space of possible x is finite. For example, say the number of possible x_i is so large that there are many values $x_i = a$ that never occurred in the most recent block of Monte Carlo samples. So one way to fill in $E(G \mid x_i = a)$ for those values is to use supervised learning to generalize from the pairs $\{(x_i \neq a, G(x)\}$ that *did* occur in the data from that block. At the end of every such block, conventional (non-sample-correcting) approaches would require that for every a we evaluate and then update $q_i(a)$. This is not needed if one uses sample correction however.

8.2. *Single agent sample correction*

Sample correction can be used even without product distributions, simply by having the number of "agents" equal 1, with the full joint variable x being that single agent's move. This can be done for either finite or infinite number of possible x's.

In the scenario discussed above where there are multiple agents, the primary purpose behind having Monte Carlo blocks of multiple timesteps is to generate sample data for the agents to input into supervised learning algorithms to estimate their distributions $E(F_G \mid x_i)$. (This is step (ii) in Sec. 7.5.) Those estimated distributions are then used to help set the update ratios, which in turn govern the sample correction for the next Monte Carlo block.

However with a single agent, to evaluate update ratios one doesn't need to estimate functions like "$E(\ln(F_G) \mid x_i)$". Such estimates of values of functions are replaced with values $\ln(F_G(x))$, which are measured exactly, with no associated estimation error. So there is no need for supervised learning algorithms to acquire such conditional expectations. In general though we still need to use statistical inference to estimate quantities like $\int dx \ln(F_G(x))$ (which is the single-agent version of the more general quantity $\int dx_i E_q(\ln(F_G) \mid x_i))$ and $E_q(F_G)$. We also need such inference in general to estimate the entropy $S(q)$ and the related quantity $\int dx \ln(q(x))$.

A major potential advantage of using a single agent arises when there are strong couplings between the x_i in G. As an example, consider finding the product distribution $\prod_i q_i(x_i)$ that best approximates some distribution $p(x) \propto e^{-G(x)}$. Strong couplings in G may mean that the best possible such product distribution approximation is not very good. So if one's goal is to form an accurate approximation of such a p, using product distributions has inherent limitations. Similarly, if one is using a product distribution, strong couplings in G can cause difficulties in attaining the goals behind the other update rules discussed above.

To address such difficulties while still using multiple agents one can try using graphical models of q that are higher order than product distributions. One can also approximate terms in the Lagrangian as in the Bethe approximation, etc. Such schemes typically obviate many of the computational advantages inherent in product distributions however. Another approach, which maintains the advantages of a product distribution but can accommodate strong coupling, is to use a semi-coordinate transformation [9, 18]. Such transformations are involved and subtle exercises however.

As an alternative, one can use sample correction with a single agent. In this approach, instead of addressing the couplings in G by using a graphical model as the approximation to $e^{-G(x)}$, they are addressed in the sample correction's density estimation algorithm (distribution estimation algorithm, in the case of a finite x space). In general though, it is very straightforward to incorporate couplings between the input variables (i.e., the x_i) in density estimation algorithms. This is in contrast to the case with graphical models.

Going to a single agent doesn't affect the need for periodic density estimation, to keep the rejection frequency in the subsampling from getting too large. (Similar difficulties arise with other sample-correction algorithms.) The major potential difficulty for using a single agent and sample correction is that it may be difficult to generate an easily sampled density with a not too large rejection frequency. However consider the case of a finite space of possible x. In this scenario at the end of Monte Carlo block t one can form a product distribution $q(x) = \prod_i q_i(x_i)$ where the values of each $q_i(x_i)$ are estimated by frequency counts on the (kept) samples. Those samples were formed by exact sampling of $p^t(x)$. Accordingly the $q_i(x_i)$ are unbiased estimates of the marginals of $p^t(x)$. In turn, the product of marginals is exactly the product distribution with minimal pq KL distance to p^t.

So in this scheme each q_i in the density estimate is set exactly as in the conventional many-agent PC approach of minimizing pq KL distance to a target distribution. However now that density estimate of p^t is corrected via sample correction.[y]

Note how much the subsampling approach simplifies in this scenario. Typically the updating of the density estimate is all that changes as the algorithm generates more data. There is no computational sense in which one updates p^t between updates of the density estimate; one is simply generating more samples.

[y]It is illuminating to compare this scheme to other schemes that interleave keep/reject steps and multi-agent updating of product distributions, e.g., those described in Sec. 3.1.

If the subsampling in this scheme results in a high rejection frequency, then our q is a poor fit to p^t. In such a case the conventional PC approach with many agents would be expected to give a product distribution that poorly approximates the target distribution. Here that shortcoming of a poor approximation doesn't hold; instead though one has the shortcoming of many rejections in the subsampling, and therefore must update the density estimate.

Note that with subsampling there is a natural way to control the degree to which we're using a (sample corrected) single agent versus a set of independent (product distribution) agents. This is done by multiplying every rejection probability by a constant $k \in [0, 1]$ before deciding whether to keep or reject a candidate sample point. $k = 1$ is full sample correction of the update rules, and $k = 0$ corresponds to no correction at all, i.e., to just using the proposal distribution. In particular, consider the case where there is a single agent but the proposal distribution is a product distribution. For this case $k = 0$ means the update rules are being used as the conventional manner to update product distributions. In contrast $k = 1$ means we are using them to update the single agent distribution.

To illustrate this, recall the update rule for iterative focusing of $\tilde{q}$ by minimization of pq distance and a Heaviside focusing function, $\Theta(G < K)$. For this focusing function the update rule Eq. 33 reduces to $q_i^{t+1}(x_i) \propto \tilde{q}(x_i \mid G < K)$, where for simplicity we can take $\tilde{q} = q^t$, the current distribution. In conventional iterative focusing based on this rule we form a set of samples of q^t. Of those we only keep the ones with $G < K$. We then use those kept samples to estimate each of the $q_i^{t+1}(x_i)$, using regression or (in the case of countable x) simple bin-counts.

Now consider how things change if we use subsampling based on a single agent, for the same update rule of Eq. 33. Now our perspective changes; we view the product distribution at time t as $\hat{q}^t$, our density estimate of the actual desired distribution $\tilde{q}$. In other words, it is now a proposal distribution. We start by forming a sample x of this distribution, and reject x if $G(x) \geq K$, just like in conventional iterative focusing. Next though we flip a biased coin, with a bias based on the ratio $\hat{q}^t(x)/q^t(x)$. We then keep x only if that coin comes up positive. Then we restart the process to get a new sample point. After collecting a large number of points this way, we use them to update our density estimate, i.e., to estimate each of the $q_i^{t+1}(x_i)$. We do this using the exact same regression or bin-counting scheme used in conventional iterative focusing.

The only difference between this and conventional use of iterative focusing with product distributions is that with subsampling, we interject a step, of flipping a biased coin. Accordingly, we can multiply the rejection bias of that coin by some factor τ to tune between the two schemes. $\tau = 0$ corresponds to conventional iterative focusing, and $\tau = 1$ is subsampling. Intermediate τ trade off the efficiencies of the two algorithms.

Note that with a single agent the Maxent Lagrangian (for example) is a convex function of one's distribution. Its minimum is interior to the feasible region, lying exactly at the desired p. This provides formal guarantees that are absent if one does not use a single (sample-corrected) agent. In addition the distribution that "best KL approximates a (distribution) function of G" is an exact fit to that function. This is true whether one uses qp or pq KL distance [8]. Accordingly "Nearest Newton" is now exactly Newton descent, with no error introduced by a last step of setting q to minimize the pq KL distance to the desired distribution. Moreover, many of the other update rules now become identical. For example Brouwer updating becomes the same as minimizing pq distance.

Furthermore, the fact that KL-based fits are exact with a single agent means that by using a single agent the guarantees of Prop. 1 now apply to iterative focusing. So we are formally guaranteed (up to sampling noise issues) that each iteration of iterative focusing lowers $E(G)$. (No such guarantees hold if one does not use sample correction.)

Moreover, consider using a single agent and iterative focusing with a Boltzmann focusing function. In this situation, the focusing step becomes identical to annealing the temperature in the parallel Brouwer update rule; iterative focusing update rules becomes identical to update rules based on the Maxent Lagrangian. Note that this algorithm relies at its core on forming samples from the proposal distribution $\hat{q}$. There is no sense in which this scheme could be used without such samples, by evaluating expressions in closed form and using that to update some variables. This contrasts with direct application of an update rule to an explicitly stored q, without any use of subsampling. Such direct application of the update rules can theoretically be done without any Monte-Carlo sampling at all. This is done by evaluating terms like $E(G \mid x_i)$ directly, in closed form, from knowledge of q. (See [18] for an example of doing this in practice.)

Finally, say we are given a distribution $p^*(x)$ that we can evaluate for any x via a black-box algorithm of some sort. Say we want to form a set of IID samples of p^*. Traditionally one could use a scheme like Metropolis-Hastings to do this. Subsampling with a single agent provides an alterative approach. This alternative starts by defining $G(x) \equiv -\ln[p^*(x)]$. So p^* is just a Boltzmann distribution over values of G. Accordingly, if we could find a (single-agent) q that minimizes KL distance to a Boltzmann distribution over G for $\beta = 1$, and generate IID samples of that q, we would have our desired IID samples of p^*. This goal is exactly met if we use subsampling with a single agent for an update rule based on KL distance to a Boltzmann distribution, once that update rule reaches equilibrium for $\beta = 1$. (Note that at that minimum the KL distance to the target distribution — which happens to equal p^* — is just 0.)

8.3. *Bootstrapping with a single agent*

Recall the discussion of forming an infinite number of bootstrap passes through the subsampling procedure, discussed at the end of the previous section. In particular recall the discussion about that limit in the case of finite X. Note that if we have a single agent, then that infinite-pass bootstrap procedure is *exactly* the same as adaptive importance sampling. The $\tilde{q}$ distribution in adaptive importance sampling is the density estimate $\hat{q}$ in the subsampling procedure. So bootstrapping is intimately related to the pq distance.

However whereas it isn't clear how to do adaptive importance sampling for uncountable X, we can always perform multiple bootstrap subsampling passes through a (single agent) set of Monte Carlo samples, and then average the resultant distribution updates. Similarly, we can perform multiple bootstrap passes through the Monte Carlo samples when our updating of q is based on a qp Lagrangian modified to incorporate X constraints.

9. Conclusion

Recent work has shown how information theory extends conventional full-rationality game theory to allow bounded rational agents. The associated mathematical framework can be used to solve distributed optimization and control problems. This is done by translating the distributed problem into an iterated game, where each agent's mixed strategy (i.e., its stochastically determined move) sets a different variable of the problem. So the expected value of the objective function of the distributed problem is determined by the joint probability distribution across the moves of the agents. The mixed strategies of the agents are updated from one game iteration to the next so as to converge on a joint distribution that optimizes that expected value of the objective function.

In this paper a set of new techniques for this updating is presented. These and older techniques are then extended to apply to uncountable move spaces. We also present an extension of the approach to include (in)equality constraints over the underlying variables. Another contribution is that we how to extend the Monte Carlo version of the approach to cases where some agents have no Monte Carlo samples for some of their moves, and derive an "automatic annealing schedule".

Future work will involve a huge number of topics. Among them are computer experiments investigating the many PC update rules and associated optimization scenarios that have yet to be empirically explored. These include experiments on constrained optimization and continuous spaces, as well as experiments when x consists of a time-extended trajectory ([8]), experiments on mixed data type move spaces, etc. Other topics for research include modifications to PC to address restrictions on the private utilities of the agents arising due to communication limitations (so that not all agents can evaluate the world utility $G(x)$ in full for each Monte Carlo sample x). Still others, closely related to economics, involve the dynamic determination of hierarchical "optimal organization charts" governing the order and groupings of agents that simultaneously update their distributions. (This is based on partial parallel-serial Brouwer updating in particular [14].) Other future work involves non-blind agents, i.e., agents that get more than just $G(x)$ values back with each Monte Carlo sample.

Acknowledgments

CEMS and DGR were funded by the US Dept. of Energy's Genomics: GTL program under project, "Carbon Sequestration in Synechococcus Sp.: ¿From Molecular Machines to Hierarchical Modeling" (www.genomes-to-life.org), US Dept. of Homeland of Homeland Security under "Next Generation Assays" and the Los Alamos LDRD program.

References

[1] D. H. Wolpert, "Product distribution field theory," 2003, preprint cond-mat/0307630.
[2] ——, "Information theory - the bridge connecting bounded rational game theory and statistical physics," in *Complex Engineering Systems*, D. Braha and Y. Bar-Yam, Eds., 2004.
[3] D. H. Wolpert, K. Tumer, and E. Bandari, "Improving search by using intelligent coordinates," *Physical Review E*, vol. 69, no. 017701, 2004.
[4] D. H. Wolpert and K. Tumer, "Optimal payoff functions for members of collectives," *Advances in Complex Systems*, vol. 4, no. 2/3, pp. 265–279, 2001.
[5] S. Bieniawski, I. Kroo, and D. H. Wolpert, "Flight Control with Distributed Effectors," in *Proceedings of 2005 AIAA Guidance, Navigation, and Control Conference, San Francisco, CA*, 2005, AIAA Paper 2005-6074.
[6] W. Macready, S. Bieniawski, and D. Wolpert, "Adaptive multi-agent systems for constrained optimization," 2004, technical report IC-04-123.
[7] C. F. Lee and D. H. Wolpert, "Product distribution theory and semi-coordinate transformations," *Proc. of the Third Intl. Conf. on Autonomous Agents and Multiagent Systems*, pp. 522–529, 2004.
[8] D. H. Wolpert and S. Bieniawski, "Distributed Control by Lagrangian Steepest Descent," 2004, pp. 1562–1567.
[9] ——, "Adaptive distributed control: beyond single-instant categorical variables," in *Proceedings of MSRAS04*, A. Skowron, Ed. Springer Verlag, 2004.
[10] S. Bieniawski and D. H. Wolpert, "Adaptive, distributed control of constrained multiagent systems," in *Proc. of the Third Intl. Conf. on Autonomous Agents and Multiagent Systems*, 1230-1231.
[11] S. Bieniawski, D. H. Wolpert, and I. Kroo, "Discrete, continuous, and constrained optimization using collectives," in *Proceedings of 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Albany, New York*, 2004.
[12] N. Antoine, S. Bieniawski, I. Kroo, and D. H. Wolpert, "Fleet assignment using Collective Intelligence," in *Proceedings of 42nd Aerospace Sciences Meeting*, 2004, AIAA-2004-0622.
[13] D. H. Wolpert and C. F. Lee, "Adaptive Metropolis-Hastings sampling using product distributions," 2004, cond-mat/0504163.
[14] D. H. Wolpert, "What Information theory says about best response, binding contracts, and Collective Intelligence," in *Proceedings of WEHIA04*, A. Namatame, Ed. Springer Verlag, 2004.
[15] D. H. Wolpert, K. Tumer, and J. Frank, "Using collective intelligence to route internet traffic," in *Advances in Neural Information Processing Systems - 11.* MIT Press, 1999, pp. 952–958.
[16] D. H. Wolpert and K. Tumer, "Collective Intelligence, Data Routing and Braess' Paradox," *Journal of Artificial Intelligence Research*, vol. 16, pp. 359–387, 2002.
[17] D. H. Wolpert, "Theory of Collective Intelligence," in *Collectives and the Design of*

Complex Systems, K. Tumer and D. H. Wolpert, Eds. New York: Springer, 2003.
[18] W. Macready and D. H. Wolpert, "Distributed constrained optimization with semi-coordinate transformations," 2005, submitted to Journal of Operations Research.
[19] J. R. Meginniss, "A new class of symmetric utility rules for gambles, subjective marginal probability functions, and a generalized Bayes' rule," *Proc. of the American Statisticical Association, Business and Economics Statistics Section*, pp. 471 – 476, 1976.
[20] D. Fudenberg and D. K. Levine, *The Theory of Learning in Games.* Cambridge, MA: MIT Press, 1998.
[21] J. Shamma and G. Arslan, "Dynamic fictitious play, dynamic gradient play, and distributed convergence to nash equilibria," *IEEE Trans. on Automatic Control*, vol. 50, no. 3, pp. 312–327, 2004.
[22] E. T. Jaynes, "Information theory and statistical mechanics," *Physical Review*, vol. 106, p. 620, 1957.
[23] J. De Bonet, C. Isbell Jr., and P. Viola, "Mimic: Finding optima by estimating probability densities," in *Advances in Neural Information Processing Systems - 9.* MIT Press, 1997.
[24] R. Rubinstein and D. Kroese, *The Cross-Entropy Method.* Springer, 2004.
[25] P. Sabes and M. Jordan, "Reinforcement learning by probability matching," in *Advances in Neural Information Processing Systems - 8.* MIT Press, 1995.
[26] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2003.
[27] D. Fudenberg and D. Kreps, "Learning mixed equilibria," *Game and Economic Behavior*, vol. 5, pp. 320–367, 1993.
[28] D. Fudenberg and D. K. Levine, "Steady state learning and Nash equilibrium," *Econometrica*, vol. 61, no. 3, pp. 547–573, 1993.
[29] D. H. Wolpert and W. Macready, "Self-dissimilarity as a high dimensional complexity measure," in *Proceedings of the International Conference on Complex Systems, 2004*, 2004, in press.
[30] T. Cover and J. Thomas, *Elements of Information Theory.* New York: Wiley-Interscience, 1991.
[31] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd ed.).* Wiley and Sons, 2000.
[32] D. Mackay, *Information theory, inference, and learning algorithms.* Cambridge University Press, 2003.
[33] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press, 1998.
[34] S. Bieniawski and D. H. Wolpert, "Using product distributions for distributed optimization," in *Proceedings of ICCS 04*, 2004.
[35] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

Appendix A. Proof of validity of subsampling procedure

Subsample correction is essentially importance sampling. Formally, since the three-step sub-sampling scheme is a stochastic process, it generates x_i's according to some distribution. The following general result establishes that this distribution is the same as q^{t-1}:

Theorem 1. *Let $\mathcal{T}_1$ be the stochastic process of forming a single sample from the distribution $D(x)$, and then keeping it with probability*

$$d(x) = \frac{[p(x)/D(x)]}{M} \tag{A.1}$$

where $M \geq max_{x'}[p(x')/D(x')]$. Consider IID repeating $\mathcal{T}_1$ until we generate a total of k kept points. The probability that any particular set of k points in X are generated by this process equals the probability that they are generated by k-fold IID sampling of p.

Proof. By the definition of M, $d(x)$ is a valid probability value for all x. So consider the event of $\mathcal{T}_1$'s generating a particular point x and then keeping it. The probability of this event is $D(x)d(x)$. However this event is identical to the event of having $\mathcal{T}_1$ generate a point it ends up keeping and having that kept point equal x. Accordingly, for any x, x',

$$\begin{aligned}\frac{P(\mathcal{T}_1 \text{ generates } x \mid \mathcal{T}_1 \text{ keeps a point})}{P(\mathcal{T}_1 \text{ generates } x' \mid \mathcal{T}_1 \text{ keeps a point})} &= \frac{P(\mathcal{T}_1 \text{ generates } x, \mathcal{T}_1 \text{ keeps a point})}{P(\mathcal{T}_1 \text{ generates } x', \mathcal{T}_1 \text{ keeps a point})} \\ &= \frac{d(x)D(x)}{d(x')D(x')} \;=\; \frac{p(x)}{p(x')}.\end{aligned} \tag{A.2}$$

Therefore

$$P(\mathcal{T}_1 \text{ generates } x \mid \mathcal{T}_1 \text{ keeps a point}) = p(x) \;\; \forall x. \tag{A.3}$$

Let $\mathcal{T}_2$ be the stochastic process of performing $\mathcal{T}_1$ once if the sample point is kept, IID performing $\mathcal{T}_1$ twice otherwise. Then we have

$$\begin{aligned}&P(\mathcal{T}_2 \text{ generates } x, \mathcal{T}_2 \text{ keeps a point}) \;=\\ &\quad P(\mathcal{T}_2 \text{ generates } x \mid \mathcal{T}_2 \text{ keeps the first point }) \times P(\mathcal{T}_2 \text{ keeps the first point}) \;+\\ &\qquad P(\mathcal{T}_2 \text{ generates } x \mid \mathcal{T}_2 \text{ rejects first point and keeps the second}) \;\times\\ &\qquad\qquad P(\mathcal{T}_2 \text{ rejects first point and keeps the second})\end{aligned} \tag{A.4}$$

$$\begin{aligned}&=\\ &\quad P(\mathcal{T}_1 \text{ generates } x \mid \mathcal{T}_1 \text{ keeps the first point }) \times P(\mathcal{T}_2 \text{ keeps the first point}) \;+\\ &\qquad P(\mathcal{T}_2 \text{ generates } x \mid \mathcal{T}_2 \text{ rejects first point and keeps the second}) \;\times\\ &\qquad\qquad P(\mathcal{T}_2 \text{ rejects first point and keeps the second}).\end{aligned} \tag{A.5}$$

However

$$P(\mathcal{T}_2 \text{ generates } x \mid \mathcal{T}_2 \text{ rejects first point and keeps the second}) = p(x), \tag{A.6}$$

using the same kind of reasoning employed above to establish that $P(\mathcal{T}_1 \text{ generates } x \mid \mathcal{T}_1 \text{ keeps a point}) = p(x)$. Therefore

$$P(\mathcal{T}_2 \text{ generates } x, \mathcal{T}_2 \text{ keeps a point}) = p(x) \times \big[P(\mathcal{T}_2 \text{ keeps the first point}) + P(\mathcal{T}_2 \text{ rejects first point and keeps the second})\big]. \tag{A.7}$$

So using the same kind of ratio-based reasoning as in Eq. A.4,

$$P(\mathcal{T}_2 \text{ generates } x \mid \mathcal{T}_2 \text{ keeps a point}) = p(x). \tag{A.8}$$

By induction, we see that with the obvious definition of $\mathcal{T}_n$,

$$P(\mathcal{T}_n \text{ generates } x \mid \mathcal{T}_n \text{ rejects the first n - 1 points and keeps the n'th point}) = p(x) \tag{A.9}$$

for any $n > 0$. So define $\mathcal{T}$ as the stochastic process of IID repeating $\mathcal{T}_1$ exactly as many times as are needed to get a single kept sample point. Then since the probability that a point will eventually be kept is 1, $P(\mathcal{T}$ generates $x) = p(x)$.

Let $\mathcal{T}'$ be the process of IID repeating $\mathcal{T}$ exactly as many times as are needed to get k kept sample points, $\{x_i : i = 1, \ldots, k\}$. Since this is an k-fold IID sampling of a process that generates points x_i with probability $p(x_i)$,

$$P(\mathcal{T}' \text{ generates } \{x_i\}) = p(\{x_i\}) \tag{A.10}$$

$\forall \{x_i\}$. This completes the proof that subsample correction generates an IID sample of the target distribution. □

Note that the probability that $\mathcal{T}_1$ keeps a point equals $\int dx\; d(x)D(X) = M^{-1}$. So if we take $M = \max_{x'}[p(x')/D(x')]$ exactly, that probability of keeping a point is highest. In turn, that maximal probability is itself extremized when $D = p$. In this case all points are kept — subsampling reduces to sampling of p.